

Open source Matrix Product States: Opening ways to simulate entangled many-body quantum systems in one dimension

Daniel Jaschke,¹ Michael L. Wall,^{1,2,*} and Lincoln D. Carr¹

¹*Department of Physics, Colorado School of Mines, Golden, Colorado 80401, USA*

²*JILA, NIST and University of Colorado, Boulder, Colorado 80309-0440, USA*

Abstract

Numerical simulations are a powerful tool to study quantum systems beyond exactly solvable systems lacking an analytic expression. For one-dimensional entangled quantum systems, tensor network methods, amongst them Matrix Product States (MPSs), have attracted interest from different fields of quantum physics ranging from solid state systems to quantum simulators and quantum computing. Our open source MPS code provides the community with a toolset to analyze the statics and dynamics of one-dimensional quantum systems. Here, we present our open source library, Open Source Matrix Product States (OSMPS), of MPS methods implemented in Python and Fortran2003. The library includes tools for ground state calculation and excited states via the variational ansatz. We also support ground states for infinite systems with translational invariance. Dynamics are simulated with different algorithms, including three algorithms with support for long-range interactions. Convenient features include built-in support for fermionic systems and number conservation with rotational $\mathcal{U}(1)$ and discrete \mathbb{Z}_2 symmetries for finite systems, as well as data parallelism with MPI. We explain the principles and techniques used in this library along with examples of how to efficiently use the general interfaces to analyze the Ising and Bose-Hubbard models. This description includes the preparation of simulations as well as dispatching and post-processing of them.

Keywords: many-body quantum system; entangled quantum dynamics; Matrix Product State (MPS); quantum simulator; tensor network method; Density Matrix Renormalization Group (DMRG)

* Present address: The Johns Hopkins University Applied Physics Laboratory, Laurel, MD, 20723, USA

Program summary

Program title: Open Source Matrix Product States (OSMPS), v2.0

Program summary and documentation: <http://openmps.sourceforge.io/>

Program obtainable from: <http://sourceforge.net/p/openmps>

Licensing provisions: GNU GPL v3 (Minor parts follow the copyright of the Expokit package.)

Programming language: Python, Fortran2003, MPI for parallel computing

Compilers (Fortran): gfortran, ifort, g95

Operating system: Linux, Mac OS X, Windows

Supplementary material: We provide programs to reproduce selected figures in the Appendices.

Nature of the problem: Solving the ground state and dynamics of a many-body entangled quantum system is a challenging problem; the Hilbert space grows exponentially with system size. Complete diagonalization of the Hilbert space to floating point precision is limited to less than forty qubits.

Solution method: Matrix Product States in one spatial dimension overcome the exponentially growing Hilbert space by truncating the least important parts of it. The error can be well controlled. Local neighboring sites are variationally optimized in order to minimize the energy of the complete system. We can target the ground state and low lying excited states. Moreover, we offer various methods to solve the time evolution following the many-body Schrödinger equation. These methods include e.g. the Suzuki-Trotter decompositions using local propagators or the Krylov method, both approximating the propagator on the complete Hilbert space.

CONTENTS

I. Introduction	3
II. Basic concepts in tensor network techniques	5
III. Defining systems and variational ground state search	10
A. Operators	11
B. Hamiltonians	12
C. Observables	14
D. Fundamentals of the library: Variational ground state search	15
E. Running the simulations	18
IV. Highlights of static algorithms	19
A. Excited state search	20
B. Infinite systems in the thermodynamic limit	21
V. Time evolution methods	23
A. Computational error and convergence	23
B. Krylov time evolution	26
C. Sornborger-Stewart decomposition	27
D. Time-dependent variational principle	28
E. Local Runge-Kutta propagation	29
F. Time evolution case study: Bose-Hubbard model in a rotating saddle point potential	29

VI. Future developments	31
VII. Conclusions	32
Acknowledgments	33
References	33
Appendices	37
A. Convenient features	37
B. Convergence studies	41
1. Finite size variational algorithms	41
2. Time evolution methods for finite size systems	43
C. Scaling of computational resources	45
D. Building and installing the open source Matrix Product States library	47
E. User support and contributing to the code	47
F. Error bounds for static simulations	48
1. Bounding ϵ with the variance delivered by open source Matrix Product States	48
2. Bounding observables	50
3. Density matrices and their bounds	51
4. Bound for the trace distance	52
5. Bound on the bond entropy	53
G. Bounding measurement with the trace distance	53
H. Details of the Krylov method	54
I. Auxiliary calculations	55
J. Files to reproduce plots in this work	56

I. INTRODUCTION

Numerical methods have been widely used to study physical systems in quantum mechanics that are not exactly solvable. In many-body systems we encounter with the exponentially growing Hilbert space a challenge to develop methods which can still simulate quantum systems on a classical computer. Starting with the Density Matrix Renormalization Group (DMRG) [1, 2], a wide range of tensor network methods have been developed. Especially in one dimension, where numerical scaling and conditioning are best, such methods offer strong alternatives to other methods such as Quantum Monte Carlo [3, 4] and the Truncated Wigner approximation [5, 6]. Applications include solid state systems, ultracold atoms and molecules, Rydberg atoms, quantum information and quantum computing, and Josephson junction-based superconducting electro-mechanical nano devices. Matrix Product States (MPSs) [7–10] define the tensor network at the foundation of the

DMRG method. MPSs themselves represent a pure quantum state constructed on local Hilbert spaces of lattice sites or discretized systems. They handle the exponentially growing Hilbert space by limiting the entanglement between any two parts of the system. Numerical methods using MPSs support static results such as ground states and time evolution of pure states. In principle, highly entangled states can be represented as MPSs, but due to the upper bound of entanglement set as a parameter, they are only accurate as long as the entanglement does not exceed this bound guaranteeing feasible computation times. This point is addressed in the main part of this paper in detail. Moreover, MPSs can exploit intrinsic characteristics of the systems such as symmetries [11, 12]. Beyond MPSs, tensor network methods are extended for multiple purposes such as 2D systems via Projected Entangled Pair States (PEPS) [13], tree tensor networks (TTNs) [14], open systems with quantum trajectories (QT) [15, 16] or Matrix Product Density Operators (MPDOs) [17, 18]. Although there have been many developments in many-body quantum simulation over the last ten to twenty years [19], from multiscale entanglement renormalization ansatz (MERA) [20] to minimally entangled typical thermal states (METTS) [21] to dynamical mean-field theory [22] to time-dependent density functional theory (TDDFT) [23, 24], MPS methods are the most often used and well-established for strongly correlated quantum systems and appropriate for large-scale open source development. The impact of these methods is represented by the large number of open source packages for MPS and DMRG [25–38], not counting proprietary efforts of multiple other groups.

We present in this paper our open source Matrix Product State (OSMPS) library, which is available on SourceForge [39]. We have over 2300 downloads since its initial release in January 2014. The library or a derivative of the library has been used in various publications [40–55]. Our implementations cover features such as variational ground and excited state searches and real time evolution for finite systems as well as ground states of infinite systems [56]. We provide built-in features such as support for symmetries, e.g. rotational $\mathcal{U}(1)$ symmetry used for number conservation in the Bose-Hubbard model and discrete \mathbb{Z}_2 symmetry occurring in the quantum Ising model, and present them in the case studies of this article. These symmetries lead to a speedup in terms of computation time and allow us to address specific states. Our libraries also support data parallel execution via Message Passing Interface (MPI) to utilize modern high performance computing resources efficiently. We illustrate the algorithms in our library together with examples of models.

One motivation for the development of OSMPS is our focus on ultracold molecules and other quantum simulator architectures incorporating long-range interacting synthetic quantum matter. Where some MPS-based algorithms are limited to nearest neighbor terms in one-dimensional systems, molecules and many other systems have long-range interactions, e.g. due to dipolar effects. In order to treat such systems, many of the algorithms in OSMPS, including dynamics algorithms, feature support for long-range interactions.

This paper is intended for two audiences: First, tensor network methods and our interfaces are introduced for researchers not familiar with such methods, but in need of numerical simulations of correlations, entanglement, and dynamics in many-body systems. On the other hand, experienced researchers within the tensor networks community should have a clear way to understand the concrete and useful details of our implementations. The paper is organized as follows. In Sec. II, we introduce the general idea of tensor networks, and provide in addition appropriate references for further reading. We continue with the example of the Ising model in Sec. III to demonstrate the variational ground state search including the general setup of systems and then highlight the other algorithms in the following sections. Section IV describes the variational search for excited states and the infinite MPS (iMPS) for the thermodynamic limit. The time evolution methods

including Krylov, Time-Evolving Block Decimation (TEBD), Time-Dependent Variational Principle (TDVP), and local Runge-Kutta follow in Sec. V. We describe future developments ahead of the conclusion in Sec. VII. The appendices cover topics such as convergence studies for the algorithms, convenient features, and technical information for installation and the scope of the open source project.

II. BASIC CONCEPTS IN TENSOR NETWORK TECHNIQUES

In this section, we briefly review the concepts of tensor network techniques. Readers familiar with MPS algorithms can continue on to Sec. III discussing the design of simulations specifically for OSMPS. The MPS algorithms rely heavily on the Schmidt decomposition of a quantum system, which can be explained best in the case of two subsystems 1 and 2 and their wave function $|\psi_{1,2}\rangle$. Each subsystem is defined on a local Hilbert space \mathcal{H}_k of dimension d_k , and is spanned by an orthonormal basis of states $\{|i_k\rangle\}$. The joint Hilbert space of subsystem 1 and 2 is formed via the tensor product $\mathcal{H} = \mathcal{H}_{1,2} = \mathcal{H}_1 \otimes \mathcal{H}_2$ and has a dimension $d_1 \times d_2$. The Schmidt decomposition is then based on a set of local wavefunctions $|\psi_\alpha^{[1]}\rangle$ and $|\psi_\alpha^{[2]}\rangle$

$$|\psi_{1,2}\rangle = \sum_{\alpha=1}^{\chi_{\max}} \lambda_\alpha |\psi_\alpha^{[1]}\rangle |\psi_\alpha^{[2]}\rangle. \quad (1)$$

The Schmidt decomposition corresponds to a singular value decomposition (SVD) where the singular values are the λ_α . The number of non-zero singular values serves as a coarse measure of entanglement between the systems 1 and 2, known as the Schmidt number or Schmidt rank. We dub the maximal number of singular values as χ_{\max} . Tracing out over either of the subsystems, we obtain the reduced density matrix of the other subsystem. This demonstrates that the eigenvalues of the reduced density matrices $\rho_1 = \text{Tr}_2 |\psi_{1,2}\rangle \langle \psi_{1,2}|$ and $\rho_2 = \text{Tr}_1 |\psi_{1,2}\rangle \langle \psi_{1,2}|$ are $\{\lambda_\alpha^2\}$ [57].

The Schmidt decomposition is unique up to rotations within subspaces of degenerate singular values for a chosen basis. Local unitary transformations such as basis transformations affecting each half of the bipartition separately are possible in general, because they do not change the entanglement, i.e., the number and value of non-zero singular values. MPSs generalize the Schmidt decomposition by allowing for rotations within the Schmidt bases $|\psi_\alpha^{[k]}\rangle$ that keep the amount of bipartite entanglement between the two subsystems fixed:

$$|\psi_{1,2}\rangle = \sum_{\alpha=1}^{\chi_{\max}} \sum_{i_1=1}^{d_1} \sum_{i_2=1}^{d_2} A_\alpha^{[1]i_1} A_\alpha^{[2]i_2} |i_1\rangle |i_2\rangle. \quad (2)$$

In addition, while the Schmidt decomposition is only defined for a bipartite system, the MPS form can be extended to any number of degrees of freedom.

We take the example of a two-site system to illustrate the connection between the Schmidt and MPS decompositions. Subsystem 1 is the site with $k = 1$ and subsystem two corresponds to the site with $k = 2$. The MPS decomposition of such a system is described in Eq. (2). We rewrite any state $|\psi_\alpha\rangle$ in Eq. (1) as a matrix of complex numbers $c_{i_1 i_2}$ and its corresponding basis states, that is $\sum_{i_1 i_2} c_{i_1 i_2} |i_1\rangle |i_2\rangle$. In Eq. (1), each dimension k is spanned by χ_{\max} orthonormal vectors. We can rewrite the sets of these vectors as matrices (rank-two tensors), where each column of the matrix represents a vector in the case of site 1, and each row is filled with one vector for site 2. Then, the matrix $A^{[k]}$ in Eq. (2) represents a rank-2 tensor for site k and the singular values are contained in

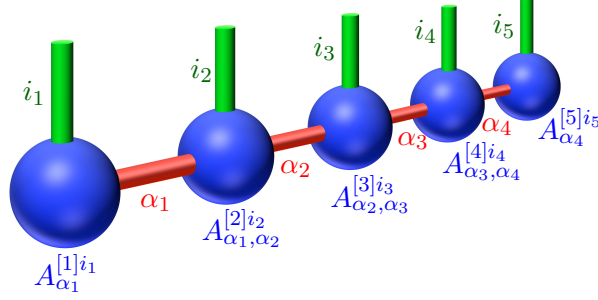


Figure 1. *Tensor network representation of an MPS of five sites for open boundary conditions.* The state is decomposed into local tensors representing each site. These tensors are connected to their nearest neighbors via the Schmidt decomposition where singular values are truncated to maintain feasible run times for larger systems. The indices for the local Hilbert space i_k are connected for measurements, e.g. for the norm $\langle \psi | \psi \rangle$, with their complex conjugated counterpart.

either $A^{[k]}$. A single element of the matrix $A^{[k]}$ can be written as $A_{\alpha}^{[k]j_k} = \langle j_k | \psi_{\alpha}^{[k]} \rangle$ in the two-site case above. Thus, the indices of the tensors $A^{[k]}$ correspond to the local Hilbert space i_k and the singular values of the Schmidt decomposition α . Throughout the paper we note the site index of a tensor in brackets, i.e the k in $A^{[k]}$.

In order to generalize this decomposition for a system with L sites, successive SVDs lead to one tensor per site where the tensors are now rank-2 at the boundaries and rank-3 in the bulk of the system, as shown in the representation as a tensor network in Fig. 1:

$$|\psi_{1,\dots,L}\rangle = \sum_{\alpha_1 \dots \alpha_{L-1}} \sum_{i_1 \dots i_L} A_{\alpha_1}^{[1]i_1} A_{\alpha_1, \alpha_2}^{[2]i_2} \dots A_{\alpha_{L-2}, \alpha_{L-1}}^{[L-1]i_{L-1}} A_{\alpha_{L-1}}^{[L]i_L} |i_1\rangle \dots |i_L\rangle. \quad (3)$$

If we allow the indices α_j to run over exponentially large values $\sim d^{L/2}$ ($d = d_k$ the local dimension, assumed uniform for simplicity), such a representation is exact, but manipulating this exact representation also scales exponentially with the system size and we do not gain anything over exact diagonalization methods.

We now introduce the key approximation in the MPS algorithm, that is the truncation of the Hilbert space according to the singular values from the Schmidt decomposition. The essential idea here is to replace the number of singular values χ_{\max} with a reduced number χ : for instance, if there are singular values less than 10^{-16} there is no reason to count them toward the amount of entanglement between the two subsystems. Thus χ becomes the *reduced Schmidt rank*, the major convergence parameter of the whole MPS algorithm, as we will show. We obtain these singular values for any splitting in two connected subsystems, and the truncation is encoded in the maximum range of the auxiliary indices α_i . Considering an approximated state $|\psi'\rangle$ truncated to the first χ singular values at some particular bond of the normalized state $|\psi\rangle$ with χ_{\max} singular values at that bond, the overlap between the two states is

$$\langle \psi' | \psi \rangle = \frac{\sum_{i=1}^{\chi} \lambda_i^2}{\sqrt{\sum_{i=1}^{\chi} \lambda_i^2}} = \sqrt{\sum_{i=1}^{\chi} \lambda_i^2}. \quad (4)$$

We prefer the overlap instead of the 2-norm of the overlap since it allows us to relate our result to the quantum fidelity in our case of pure states with real overlaps as $\mathcal{F} = \langle \psi | \psi' \rangle$. We define the

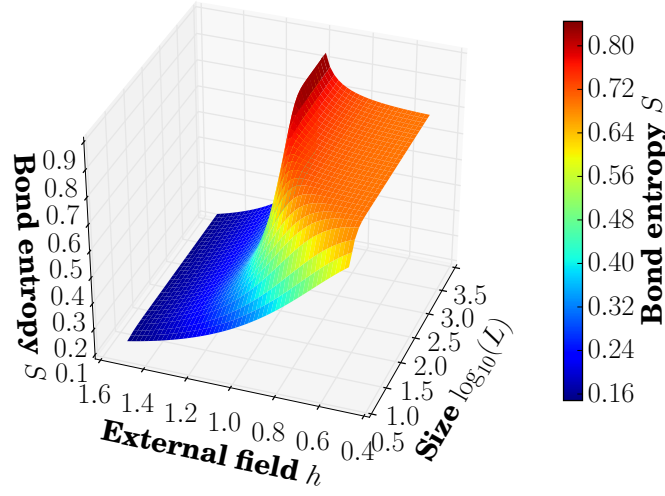


Figure 2. *The compression of the quantum state in the MPS acts on the singular values of the Schmidt decomposition in Eq. (1). The bond entropy or von Neumann entropy at the center bond for the nearest-neighbor quantum Ising model $H = -\sum \sigma_k^z \sigma_{k+1}^z - g \sum \sigma_k^x$, defined later in Eq. (15), peaks around the critical point for increasing system sizes.*

truncation error made in this step as $\epsilon_{\text{local}} = 1 - \langle \psi | \psi' \rangle$ and obtain

$$\epsilon_{\text{local}} = 1 - \sqrt{\sum_{i=1}^{\chi} \lambda_i^2} \leq \sum_{i=\chi+1}^{\chi_{\text{max}}} \lambda_i^2. \quad (5)$$

This upper bound is useful since it relates directly to the truncated singular values. Such a truncation corresponds to a truncation of entanglement through two entanglement measures: the Schmidt rank χ and the von Neumann entropy S . The former is simply the number of non-zero singular values, and is an entanglement monotone. The von Neumann entropy, or bond entropy, S is given as

$$S = -\sum_{i=1}^{\chi} \lambda_i^2 \log(\lambda_i^2). \quad (6)$$

In Fig. 2 we show the bond entropy for the bipartition at the center bond for the quantum Ising model with transverse field as a function of the system size and external field. Errors in the MPS approach originate in high entanglement; therefore, simulations for increasing system sizes and around the quantum critical point are more vulnerable to smaller cutoffs χ . The quantum critical point for the Ising model is $h_c = 1.0$ and becomes visible as a red ridge in Fig. 2 for large system sizes. The ground state of the ferromagnetic phase in the limit of zero external field, also called the Greenberger-Horne-Zeilinger (GHZ) state, is the superposition of all spins up and all spins down, i.e. $(|\uparrow \cdots \uparrow\rangle + |\downarrow \cdots \downarrow\rangle) / \sqrt{2}$. We expect an entropy of $S = -\log(0.5) \approx 0.69$, which agrees well with the results in the Fig. 2. For gapped 1D systems with short-range interactions, the so-called area law for entanglement [58] states that the entanglement at any bipartition is independent of the length of the subsystems (and hence of the system size L). Since the bond entropy is an entanglement measure, this upper bound can be used as the gap opens away from the critical point. At the critical point, the entanglement grows logarithmically with the subsystem size.

We introduce a list of basic operations that can be performed on tensor networks, and explain the orthogonality center, an isometrization or gauge, used in the OSMPS algorithms. For these linear algebra operations on tensors we suppress the basis kets of the quantum states for simplicity throughout the paper. One key feature of every MPS with open boundary conditions is that introducing an orthogonality center leads to faster local measurements and error reduction in the truncation [59]. We introduce the left and right canonical form of tensors according to

$$\begin{aligned} A_{\alpha_{k-1}, \alpha_k}^{[k]i_k} &= L_{\alpha_{k-1}, \alpha_k}^{[k]i_k} \quad \text{if} \quad \begin{cases} A^{[j]} = L^{[j]} \quad \forall j < k \\ \text{and} \quad \sum_{\alpha_{k-1}, i_k} A_{\alpha_{k-1}, \alpha_k}^{[k]i_k} (A_{\alpha_{k-1}, \alpha'_k}^{[k]i_k})^* = \mathbb{I}_{\alpha_k, \alpha'_k}, \end{cases} \\ A_{\alpha_{k-1}, \alpha_k}^{[k]i_k} &= R_{\alpha_{k-1}, \alpha_k}^{[k]i_k} \quad \text{if} \quad \begin{cases} A^{[j]} = R^{[j]} \quad \forall j > k \\ \text{and} \quad \sum_{\alpha_k, i_k} A_{\alpha_{k-1}, \alpha_k}^{[k]i_k} (A_{\alpha'_{k-1}, \alpha_k}^{[k]i_k})^* = \mathbb{I}_{\alpha_{k-1}, \alpha'_{k-1}}. \end{cases} \end{aligned} \quad (7)$$

The left (right) canonical forms $L^{[k]}$ ($R^{[k]}$) are unitary matrices, e.g. from the SVD obtained from the Schmidt decomposition in Eq. (1). These conditions apply if the singular values have not been multiplied into the tensor. We define the orthogonality center as the site which has only left orthogonal tensors on the left side and right orthogonal tensor on the right side. This feature becomes beneficial for measurements as the contractions in the condition of Eq. (7) do not have to be calculated knowing that the result is the identity \mathbb{I} . Stated equivalently, the tensor of the orthogonality center $A_{\alpha\beta}^{[k]i_k}$ consists of the coefficients of the wave function in the orthonormal basis spanned by the local states $|i_k\rangle$ and the left and right Schmidt vectors given by products of the other MPS tensors. With these definitions, we can derive the overlap from Eq. (4). We assume that we truncate singular values at the bond of the sites k and $k+1$ and the sites up to and including site k are of the form $L^{[j \leq k]}$ and the tensors beginning on site $k+1$ are of type $R^{[j \geq k+1]}$. The truncation does not affect any of the tensors $L^{[j \leq k]}$ or $R^{[j \geq k+1]}$. Contracting these tensors for the overlap with their complex conjugated counterparts, we obtain identities on all sites and $\langle \psi'' | \psi \rangle$ simplifies to

$$\langle \psi' | \psi \rangle = \sum_{\alpha_{k-1}, \alpha_k} \Lambda_{\alpha_{k-1}, \alpha_k} \Lambda'_{\alpha_{k-1}, \alpha_k}, \quad (8)$$

where $|\psi''\rangle$ is the unnormalized truncated state and $|\psi'\rangle$ the truncated normalized state. The diagonal structure of the matrices Λ containing the singular values leads to $\langle \psi' | \psi \rangle = \sum_{\alpha} \Lambda_{\alpha} \Lambda'_{\alpha}$. Since the smallest singular values in Λ' are set to zero, the result is the sum of the squared singular values in Λ' . The additional term in the denominator in Eq. (4) originates in the normalization of $|\psi''\rangle$. We emphasize that this procedure only works if the sites are completely in the form of $L^{[j \leq k]}$ and $R^{[j \geq k+1]}$, since otherwise the contraction with the complex conjugated tensor does not lead to an identity.

Moreover, we introduce the following actions on tensors in our MPS library:

- **Contractions** over two tensors are defined as the summation over one (or more) common indices, and hence generalize matrix-matrix multiplication to higher-rank tensors. A commonly used example would be to contract two neighboring tensors of an MPS, $A_{\alpha_{k-1}, \alpha_k}^{[k]i_k}$ and $A_{\alpha_k, \alpha_{k+1}}^{[k+1]i_{k+1}}$, to one tensor representing the sites k and $k+1$. The summation is in this case over the index α_k and we obtain a tensor $\Theta_{\alpha_{k-1}, \alpha_{k+1}}^{[k, k+1]i_k, i_{k+1}}$.
- **Splitting** of a tensor is the reverse action of a contraction. The indices of the tensor form two subgroups where the splitting is enacted between those two groups. Taking the two site

tensor $\Theta_{\alpha_{k-1}, \alpha_{k+1}}^{[k, k+1]i_k, i_{k+1}}$ as an example, we group α_{k-1}, i_k together and i_{k+1}, α_{k+1} in order to obtain two single site tensors, up to a possible truncation. The splitting can be achieved via three possibilities:

- An SVD splits the tensor directly into two unitary tensors and the singular values, described by

$$\Theta_{\alpha_{k-1}, \alpha_{k+1}}^{[k, k+1]i_k, i_{k+1}} = L_{\alpha_{k-1}, \alpha_k}^{[k]i_k} \Lambda_{\alpha_k} R_{\alpha_k, \alpha_{k+1}}^{[k+1]i_{k+1}}, \quad (9)$$

where the singular values Λ_{α_k} allow us to truncate the state to a certain χ . The maximal bond dimension is defined as $\min(\chi_{k-1}d_k, d_{k+1}\chi_{k+1})$.

- The eigenvalue decomposition is related to the singular value decomposition, which is the reason the eigenvalue decomposition can replace the SVD. If the SVD decomposes A into $U\Lambda V$, the eigendecomposition $\mathcal{E}(\cdot)$ is set up as follows:

$$\mathcal{E}(AA^\dagger) = \mathcal{E}(U\Lambda V V^\dagger \Lambda U^\dagger) = \mathcal{E}(U\Lambda^2 U^\dagger) = U\Lambda^2 U^\dagger. \quad (10)$$

The eigendecomposition of AA^\dagger , which is built from a matrix-matrix multiplication, returns a unitary matrix and the singular values squared. To obtain the right matrix, we multiply U^\dagger with the original matrix A leading to

$$U^\dagger A = U^\dagger U \Lambda V = \Lambda V, \quad (11)$$

which already contains the singular values. After completing the series of steps, we obtain

$$\Theta_{\alpha_{k-1}, \alpha_{k+1}}^{[k, k+1]i_k, i_{k+1}} = L_{\alpha_{k-1}, \alpha_k}^{[k]i_k} \Lambda_{\alpha_k}^{[k+1]i_{k+1}}, \quad (12)$$

where the truncation is possible due to the knowledge of Λ^2 in the intermediate step of Eq. (10), although the singular values do not appear in the previous equation (12). As in the case of the SVD, the maximal bond dimension is $\min(\chi_{k-1}d_k, d_{k+1}\chi_{k+1})$. The unitary matrix can be obtained for the right side starting with $A^\dagger A$. This procedure is generally faster than the SVD.

- The QR decomposition decomposes a matrix into a unitary matrix and an upper triangular matrix T . If the unitary matrix is on the right side, it may be referred to as RQ decomposition. It does not allow for truncation as the singular values are not calculated. The example for the QR is

$$\Theta_{\alpha_{k-1}, \alpha_{k+1}}^{[k, k+1]i_k, i_{k+1}} = L_{\alpha_{k-1}, \alpha_k}^{[k]i_k} T_{\alpha_k, \alpha_{k+1}}^{[k+1]i_{k+1}}. \quad (13)$$

Therefore, the new bond dimension is the maximal one, $\chi = \chi_{\max} = \min(\chi_{k-1}d_k, d_{k+1}\chi_{k+1})$. The fact that the QR scenario is not rank revealing is the reason for not using it in the splitting of two sites in the library, but it is used for shifting as explained in the following.

The different options for splitting a tensor are summarized in Fig. 3. We choose the SVD to obtain the singular values and two unitary matrices. In contrast, the eigenvalue decomposition yields a unitary matrix and the singular values. The QR decomposition differs from the first two methods as it does not reveal the singular values and returns only one unitary matrix. Therefore, the QR is computationally less expensive than the approach with

the eigenvalue decomposition. The SVD is computationally more costly than both other algorithms.

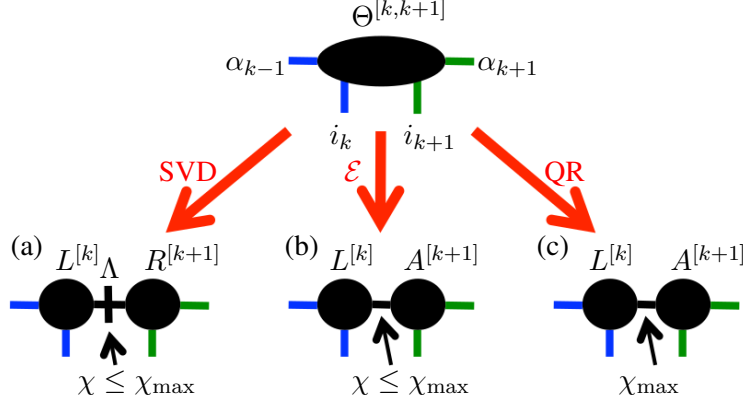


Figure 3. *Methods for splitting a two site tensor into two one site tensors include (a) an SVD decomposition, (b) an eigenvalue decomposition \mathcal{E} in combination with matrix multiplications, and (c) a QR decomposition.*

- **Shifting** the orthogonality center can be done with local operations, meaning that the operations act only at one site at a time and do not use any two site tensors. Running an SVD or QR (RQ) decomposition for site k on a single rank-3 tensor with dimensions χ_{k-1} , d , and χ_k , we reshape the tensor as a $\chi_l d \times \chi_r$ ($\chi_l \times d \chi_r$) matrix and obtain a left-canonical (right-canonical) unitary tensor for site k and an additional matrix. The additional matrix consists of the singular values contracted into a unitary matrix when choosing an SVD. For the QR (RQ) decomposition we obtain a left (right) canonical unitary and an additional upper triangular matrix. This additional matrix can be contracted to the corresponding neighboring site $k \pm 1$, resulting in that site becoming the new orthogonality center. We note in this case that the QR and RQ decomposition does not change the ranks of the matrices, and is roughly a factor of two faster than the SVD.

With the knowledge of the basic features of an MPS, we introduce in the next chapter how a model is defined in the OSMPS library and how we obtain results as in Fig. 2.

III. DEFINING SYSTEMS AND VARIATIONAL GROUND STATE SEARCH

We now outline the definition of systems in OSMPS. As an example we consider finding the ground state of the finite size quantum Ising model. The 1D long-range transverse field Ising Hamiltonian is [60, 61]

$$H = -J \sum_{i < j \leq L} \frac{\sigma_i^z \sigma_j^z}{(j-i)^\alpha} - Jh \sum_{i=1}^L \sigma_i^x, \quad (14)$$

where the operators are defined over the Pauli matrices $\{\sigma_i^x, \sigma_i^y, \sigma_i^z\}$ acting on a site i in the system. The interactions between the spins at different sites decay following a power-law introduced in the first term of the Hamiltonian governed by α , the distance $|j-i|$, and the overall energy scale J . The external field is governed by h appearing in the second term of the Hamiltonian. The number

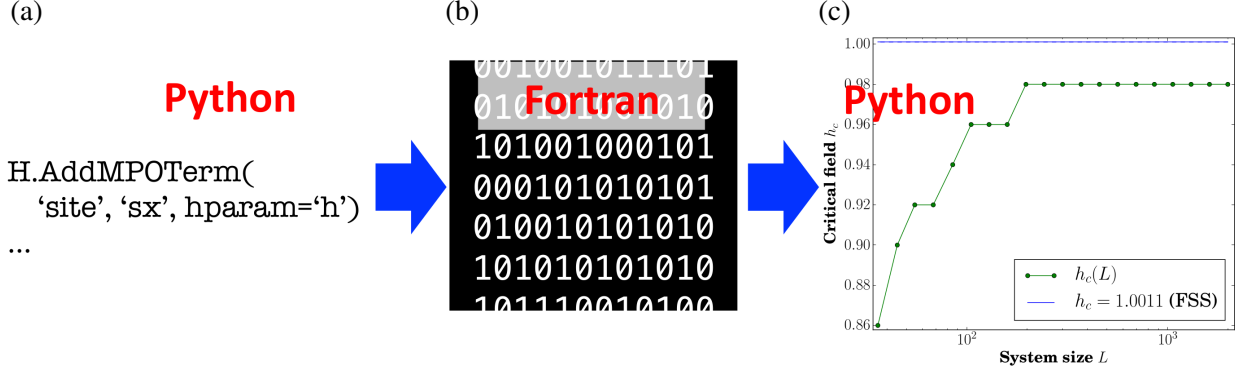


Figure 4. *OSMPS flow chart for a simulation.* The OSMPS library combines a user-friendly interface in Python with a computationally powerful core written in Fortran. (a) The simulation setup is done in Python. (b) A write function provides the files for Fortran and a corresponding read function imports the results from Fortran to Python (blue arrows). (c) The Python front end then takes care of the evaluation of the data. The plot in the flow charts shows the critical value of the external field in the Ising model as a function of the system size L evaluated via the maximum of the bond entropy. Finite size scaling (FSS) delivers the critical field in the thermodynamic limit for $L \rightarrow \infty$.

of sites is L . We focus in this section on the nearest neighbor case H_{NN} obtained for the limit $\alpha \rightarrow \infty$, commonly called the transverse quantum Ising model,

$$H_{\text{NN}} = -J \sum_{i=1}^{L-1} \sigma_i^z \sigma_{i+1}^z - Jh \sum_{i=1}^L \sigma_i^x. \quad (15)$$

The overall approach to OSMPS is a user-friendly Python environment calling a Fortran core for the actual calculations. This scheme is depicted in Fig. 4. Thus, we guide the reader through the simulation with a corresponding summary of the Python files; the complete files are contained in Appendix J.

From a quantum mechanical point of view the following steps are necessary to describe a system. First, we have to generate the operators which are acting on the local Hilbert space, described in Sec. III A. Once we have the operators, we build the Hamiltonian out of rule sets in Sec. III B. Then, we set up the measurements to be carried out in Sec. III C. This procedure completes the definition of the quantum system, but we have two more tasks with regards to the numerics. In the fourth step we define the convergence parameters of the algorithm, where Sec. III D describes this step for the variational ground state search. Finally, the simulation is set up and executed in Sec. III E.

One general comment remains before starting with a detailed description of the simulation setup. Every simulation in OSMPS is represented by a Python dictionary, which contains observables and convergence parameters as well as general parameters such as the system size.

A. Operators

OSMPS comes with predefined sets of operators for three different physical systems to facilitate the setup of simulations. These predefined sets of operators are returned by the corresponding functions for bosonic systems such as the Bose-Hubbard model, fermionic systems includ-

ing their phase operators originating from the Jordan-Wigner transformation, or spin operators. We use the last set in the example for the quantum Ising model. The corresponding function `BuildSpinOperators` returns the set of operators $\{\sigma^+, \sigma^-, S^z, \mathbb{I}\}$. In order to obtain the Pauli operators σ^x and σ^z from the spin lowering and raising operators σ^\pm and the spin operator S_z , we suggest following the prescription in Listing 1.

Listing 1. Defining the operators of the quantum Ising model overwriting S^Z with σ^Z .

```

32  # These are rotated Pauli operators to obtain a diagonal
    generator for Z2
33  Operators = mps.BuildSpinOperators(spin=0.5)
34  Operators['sx'] = 2 * Operators['sz']
35  Operators['sz'] = - (Operators['splus'] + Operators['sminus'])
36  Operators['gen'] = np.array([[0, 0], [0, 1.]])

```

B. Hamiltonians

Through these operators, stored in a dictionary-like Python class, we define the Hamiltonian and later on the observables. The Hamiltonian is described as a matrix product operator (MPO), which is effectively an MPS with rank-four tensors instead of rank-three tensors:

$$H = \sum_{i_1, \dots, i_L} \sum_{i'_1, \dots, i'_L} \sum_{\alpha_1, \dots, \alpha_{L-1}} M_{\alpha_1}^{[1]i'_1, i_1} M_{\alpha_1, \alpha_2}^{[2]i'_2, i_2} \dots M_{\alpha_{L-1}}^{[L]i'_L, i_L} |i'_1\rangle \langle i_1| \dots |i'_L\rangle \langle i_L|. \quad (16)$$

A key property is that an MPO acting on an MPS can be written as another MPS, generally with a larger bond dimension. The physical indices i_k act on the physical indices of an MPS and take them to new physical indices i'_k . The auxiliary indices α of the MPO are fused with the corresponding auxiliary indices in the MPS. For most physical operators, this structure of rank-4 tensors is sparse and therefore we rather seek for an efficient implementation in terms of MPO-matrices $M^{[k]}$ than in rank-4 tensors. The MPO matrix $M^{[k]}$ including the iteration over all indices for one site representing the rule sets [52, 62] for local terms, bond terms for the interaction of nearest neighbors, and exponential rules for long-range interactions between two sites takes the form

$$M^{[k]} = \begin{pmatrix} A_{\alpha_{k-1}=1, \alpha_k=1}^{[k]} & 0 & 0 & 0 \\ A_{\alpha_{k-1}=2, \alpha_k=1}^{[k]} & A_{\alpha_{k-1}=2, \alpha_k=2}^{[k]} & 0 & 0 \\ A_{\alpha_{k-1}=3, \alpha_k=1}^{[k]} & 0 & A_{\alpha_{k-1}=3, \alpha_k=3}^{[k]} & 0 \\ A_{\alpha_{k-1}=4, \alpha_k=1}^{[k]} & A_{\alpha_{k-1}=4, \alpha_k=2}^{[k]} & A_{\alpha_{k-1}=4, \alpha_k=3}^{[k]} & A_{\alpha_{k-1}=4, \alpha_k=4}^{[k]} \end{pmatrix}, \quad (17)$$

where $k = 2, 3, \dots, L-1$. The matrix structure in Eq. (17) corresponds to the auxiliary indices. Each element within this structure $A_{\alpha_{k-1}, \alpha_k}^{[k]}$ is a matrix acting on the local Hilbert space of site k , e.g. the Pauli matrix σ_k^z . Thus the auxiliary indices α of the rank-4 tensor encode the row and column of $M^{[k]}$, while the indices i_k and i'_k are related to the local Hilbert space are located in the rows and columns of the matrices $A_{\alpha_{k-1}, \alpha_k}^{[k]}$. We now illustrate the meaning of the matrices depending on their position in $M^{[k]}$. In order to build the MPO for the Hamiltonian for the long-range Ising model, we only need the first column, last row, and the diagonal of $M^{[k]}$ and store it as a sparse structure. Matrices in the first column (last row) of $M^{[k]}$ are multiplied with identity

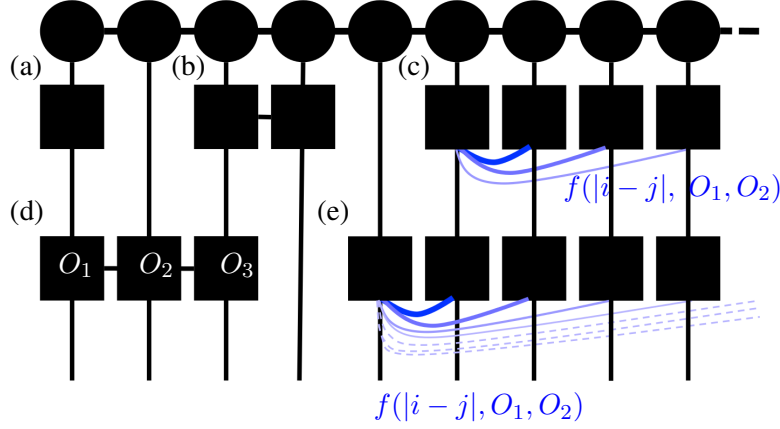


Figure 5. *Rules for building a Hamiltonian.* (a) Local terms; (b) bond terms acting on two neighboring sites; (c) finite range terms of two operators O_1, O_2 and a coupling depending on the distance. The coupling function $f(x)$ is a finite sum over a limited number of neighboring sites. (d) String of arbitrary operators, e.g., a three-body term built from O_1, O_2 , and O_3 and (e) infinite terms of two operators O_1 and O_2 with a distance depending on decaying coupling either as a general function $f(|j - i|)$ (InfiniteFunction) or as an exponential (Exponential). Any infinite function is expressed as a sum of exponentials within OSMPS. The coupling function $f(x)$ is extended to all sites.

operators on the right (left) side of site k , i.e., they do not interact with any sites right (left) of themselves. Diagonal elements propagate operators through the system and are completed by other operators to the left and right of site k , and hence represent long-range interactions. For the first and last site we define the MPO-matrix as vectors

$$M^{[1]} = \begin{pmatrix} A_{1,1}^{[1]} & A_{1,2}^{[1]} & A_{1,3}^{[1]} & A_{1,4}^{[1]} \end{pmatrix}, \quad M^{[L]} = \begin{pmatrix} A_{1,1}^{[L]} \\ A_{2,1}^{[L]} \\ A_{3,1}^{[L]} \\ A_{4,1}^{[L]} \end{pmatrix}, \quad (18)$$

corresponding to the auxiliary rank-one structure for the MPO matrices at the boundary in Eq. (16). Note that the first MPO matrix is a row vector and the last is a column vector, resulting in the contracted MPO object Eq. (16) being a 1×1 matrix in the auxiliary indices.

We now build the nearest neighbor Hamiltonian H_{NN} from Eq. (15) for the quantum Ising model to continue with our example. Figure 5 shows the possible rule sets provided through OSMPS. We need the local site term depicted in Fig. 5(a) and the bond term from Fig. 5(b). In general these operators are filled with identities on all other sites and act on each possible site. The corresponding MPO-matrices depending on the site index k are then

$$M^{[1]} = \begin{pmatrix} -h\sigma_1^x & -J\sigma_1^z & \mathbb{I} \end{pmatrix}, \quad M^{[k]} = \begin{pmatrix} \mathbb{I} & 0 & 0 \\ \sigma_k^z & 0 & 0 \\ -h\sigma_k^x & -J\sigma_k^z & \mathbb{I} \end{pmatrix}, \quad M^{[L]} = \begin{pmatrix} \mathbb{I} \\ \sigma_L^z \\ -h\sigma_L^x \end{pmatrix}, \quad (19)$$

where $k = 2, 3, \dots, L - 1$. To see how this MPO structure results in the proper many-body operator, we will explicitly build the Hamiltonian for three sites, i.e., $H = M^{[1]} \times M^{[2]} \times M^{[3]}$, where \times is understood to be ordinary matrix multiplication in the auxiliary indices together with

tensor products on the physical indices. We start by multiplying the row vector for the first site with the matrix for the second site

$$M^{[1]} \times M^{[2]} = \begin{pmatrix} -h\sigma_1^x & -J\sigma_1^z & \mathbb{I} \end{pmatrix} \times \begin{pmatrix} \mathbb{I} & 0 & 0 \\ \sigma_2^z & 0 & 0 \\ -h\sigma_2^x & -J\sigma_2^z & \mathbb{I} \end{pmatrix} = \begin{pmatrix} (-h\sigma_1^x - J\sigma_1^z\sigma_2^x - h\sigma_2^x) & -J\sigma_2^z & \mathbb{I} \end{pmatrix}. \quad (20)$$

The multiplication of this result with the column vector for the last and third site results then in the Hamiltonian

$$\begin{aligned} M^{[1]} \times M^{[2]} \times M^{[3]} &= \begin{pmatrix} (-h\sigma_1^x - J\sigma_1^z\sigma_2^x - h\sigma_2^x) & -J\sigma_2^z & \mathbb{I} \end{pmatrix} \times \begin{pmatrix} \mathbb{I} \\ \sigma_3^z \\ -h\sigma_3^x \end{pmatrix} \\ &= -h\sigma_1^x - J\sigma_1^z\sigma_2^x - h\sigma_2^x - J\sigma_2^z\sigma_3^z - h\sigma_3^x. \end{aligned} \quad (21)$$

The MPO matrices have more entries for models beyond nearest neighbor interactions. The local terms remain in the last row of the first column as the identities stay in their places. The diagonal is set in the case of long-range interactions with an identity times a decay factor. The larger the distance between two sites becomes, the higher the contribution of the decay multiplied at each site in between. Elements in the lower triangular part of the matrix besides the first column and last row are used e.g. in the `FiniteTerm` rule set.

Independent of the system, we first initialize an instance of the `MPO` class. The different types of terms are specified via a string argument in the class function `AddObservable`. Keyword arguments to any MPO terms are the weight and Hamiltonian parameters `hparam`. Further arguments specific for the rule can be found in the documentation, e.g. the infinite function can take the function as an additional keyword argument.

Listing 2. Defining the Hamiltonian of the quantum Ising model.

```

41 H = mps.MPO(Operators)
42 H.AddMPOTerm('bond', ['sz', 'sz'], hparam='J', weight=-1.0)
43 H.AddMPOTerm('site', 'sx', hparam='h', weight=-1.0)

```

In the code we have given a string variable name for the coupling of the bond and site term. The energy scale ($J=1$) and the different values for h are specified later in the Python setup script inside the dictionary representing the simulation allowing for flexibility.

C. Observables

In order to evaluate the behavior of the system, we have to define the observables. Figure 6 shows the possible measurements: local site terms including site and bond entropy, correlations, MPOs, string operators and one or two-site reduced density matrices where the reduced density matrices ρ_i and $\rho_{i,j}$ are defined as

$$\rho_i = \text{Tr}_{k \neq i}(\rho), \quad \rho_{i,j} = \text{Tr}_{k \neq i,j}(\rho), \quad (22)$$

where the density matrix on the complete system is defined as $\rho = |\psi\rangle\langle\psi|$. The energy as an MPO measurement of the Hamiltonian, the bond dimension, the variance within variational algorithms,

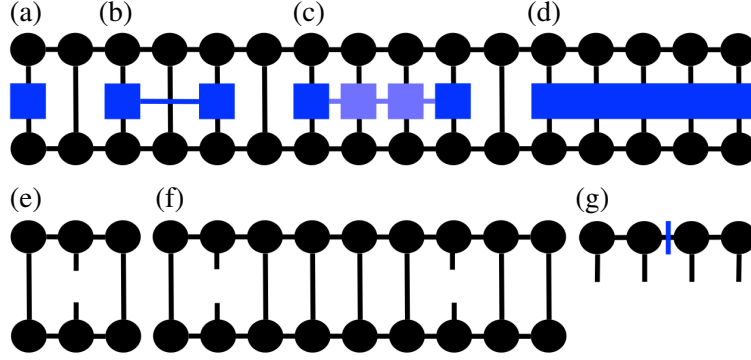


Figure 6. *OSMPS measurements* can be selected from the following options: (a) Local terms. (b) Two-site correlators including correlations for fermionic systems. (c) String operators of type $\langle O_A O_B O_B \dots O_B O_B O_C \rangle$. (d) MPO as used in the default measurement of the energy (Hamiltonian). (e) Single site density matrices tracing over the remaining system. (f) Two-site density matrices tracing out everything but two sites as defined in Eq. (22). (g) Singular values between left and right part of the MPS.

or the overlap between the initial state and the time evolved state (Loschmidt echo) are measured by default. For the Ising example, we measure $\langle \sigma_i^z \rangle$ and $\langle \sigma_i^z \sigma_j^z \rangle$. Due to the local observable we gain as well the bond entropy shown in Fig. 2. The following Listing 3 shows the necessary code for measuring these observables.

Listing 3. Defining the observables of the quantum Ising model.

```

60     # Initialize instance of observable class and add local
        observable
61     myObservables = mps.Observables(Operators)
62     myObservables.AddObservable('site', 'sz', 'z')
63
64     # add correlation functions
65     myObservables.AddObservable('corr', ['sz', 'sz'], 'zz')

```

D. Fundamentals of the library: Variational ground state search

The previous steps completed the setup of the physical system, and we continue with the specification of the convergence parameters. Therefore, we explain the variational algorithm used to find the ground state which serves as input for the algorithms for excited state search and real time evolution.

From exact diagonalization, we know how to find the ground state via solving the eigenequation, which is optimally done with sparse methods such as the Lanczos algorithm [63]. The same procedure cannot be used in the same way beyond a few tens of particles due to the exponentially growing Hilbert space in the many-body system, but the variational ground state search adapts the eigenvalue problem to an effective eigenvalue problem for a few neighboring sites. In principle, the eigenequation $H|\psi\rangle = E|\psi\rangle$ can be solved for the ground state on the complete Hilbert space

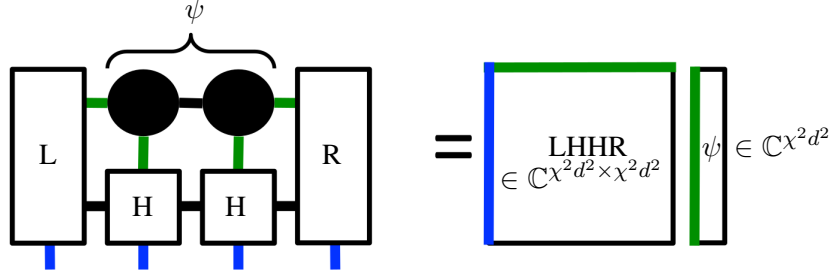


Figure 7. *Effective Hamiltonian for the Lanczos algorithm* is built via the contractions of all MPO matrices with tensors for the sites $k' \neq k, k + 1$.

using imaginary time evolution, e.g. with the Krylov method presented for the dynamics later, using the equation $e^{-\beta H}$ with $\beta \rightarrow \infty$. The thermodynamic beta approaches infinity as the system approaches the ground state at zero temperature. Instead of searching for the global minimum, we seek for local minima transferring the problem to an effective eigenequation for n neighboring sites in the OSMPS algorithm. The other $L - n$ sites are kept fixed while finding the effective ground state of the n sites. This effective eigenproblem does not grow exponentially with the system size, but depends on the local dimension and the bond dimension of the constant parts of the system, i.e. $d^n \chi^2$. The number of these effective eigenvalue problems grows linearly with the system size. In the following for simplicity we set $n = 2$, which corresponds to the value used in OSMPS:

$$\varepsilon[|\psi\rangle] = \langle\psi|H|\psi\rangle - E\langle\psi|\psi\rangle \quad \underset{\text{global}}{\overset{\text{local}}{\rightleftharpoons}} \quad H_{\text{eff}} A^{[k,k+1]} = E A^{[k,k+1]}. \quad (23)$$

The local minimization over n neighboring sites is done iteratively moving through the neighboring pairs of sites until convergence is reached (see details in Appendix B). We point out the role of the effective Hamiltonian in more detail with regards to the Lanczos algorithm. The Lanczos algorithm finds the eigenvalues and vector of a problem using only matrix vector multiplications, in our case $H|v\rangle$ for some vector $|v\rangle$. Because we restrict ourselves to the sites $k, k + 1$, the tensors of the other sites remain constant and we can contract them with their MPO matrices. These fixed sites form an environment which acts as part of the total matrix vector multiplication. The contraction can be continued until we only have one tensor L to the left and one tensor R to the right representing those contractions.¹ Together with the MPO matrices $M^{[k]}$ and $M^{[k+1]}$, the tensors L and R build the effective Hamiltonian. We now find the minimum in energy for this effective Hamiltonian with regards to sites k and $k + 1$. This effective Hamiltonian is resumed in Fig. 7. In the case of matrices, the Lanczos algorithm is ideal for sparse problems and calculating only a few eigenvectors. In the tensor network scenario it provides a considerable speedup in contrast to dense methods due to the tensor network structure: contracting the tensors L , R , and the MPO matrices $M^{[k]}$, and $M^{[k+1]}$, step-by-step to $|v\rangle$ is more efficient than building H_{eff} of dimension $\chi^2 d^2 \times \chi^2 d^2$ and multiplying it with $|v\rangle$ or solving the eigenvalue problem, i.e. $\mathcal{O}(\chi^3)$ versus $\mathcal{O}(\chi^6)$ [52]. The two-site eigenvalue problem corresponds to finding the stationary point of the energy functional through the equation

$$\frac{\partial}{\partial (A^{[k,k+1]})} (\langle\psi|H|\psi\rangle - E\langle\psi|\psi\rangle) = 0. \quad (24)$$

¹ In this context the symbol L represents the left tensor and not the system size.

where E , the energy eigenvalue, is a Lagrange multiplier enforcing normalization, and the derivative with respect to a tensor is defined to be a tensor of the same shape whose elements are the derivatives with respect to the individual tensor elements.

Listing 4. Defining the two sets of convergence parameters for the quantum Ising simulation.

```

71     conv = mps.MPSConvParam(max_bond_dimension=40, variance_tol=1e
      -10,
72                               local_tol=1e-10, max_num_sweeps=4)
73     conv.AddModifiedConvergenceParameters(0, ['max_bond_dimension',
74                                               'max_num_sweeps'],
      [80, 4])

```

The key to obtaining meaningful results are the convergence parameters of the variational algorithm. The convergence parameters are stored in a corresponding Python object which is shown in Listing 4 and the different parameters are defined in the following. For example, the variance $V_\psi = \langle H^2 - \langle H \rangle^2 \rangle$ indicates the distance from an eigenstate. Table I presents out of the analysis in Appendix B the parameters to obtain ground states with a variance tolerance $\varepsilon_V = 10^{-12}$, effectively $L \times 10^{-12}$ for the whole system, for different models. Here, we concentrate on the values of the Lanczos tolerance ε_l and bond dimension where other parameters are kept constant. Those are especially interesting because the bond dimension defines the fraction of the Hilbert space which can be captured. On the other hand, the Lanczos tolerance determines the accuracy of the eigenvector solved in Eq. (24). The parameters kept constant are the number of Lanczos iterations. If the number of Lanczos iterations is sufficiently high the accuracy of the Lanczos tolerance is met, otherwise not. The local tolerance $\varepsilon_{\text{local}}$, defining the cutoff of the singular values in the Schmidt decomposition of Eq. (1), guarantees that we do not use the full bond dimension if the sum of the singular values squared are below the local tolerance. It relates to the variance tolerance and defaults to

$$\varepsilon_{\text{local}} = \frac{\varepsilon_V}{4L}. \quad (25)$$

The motivation to choose this value is that the local error made during one sweep consists of the approximately $2L$ splittings, where the additional factor of two is a safety factor to ensure good convergence. The number of sweeps through the system, optimizing each pair of two sites twice, is specified with the number of inner sweeps N_i , which is bounded between `min_num_sweeps` and `max_num_sweeps`, and N_o , the parameter for the outer sweeps `max_outer_sweeps`. The maximal number of overall sweeps N_{sweep} is then

$$N_{\text{sweep}} = N_i \cdot N_o. \quad (26)$$

Convergence is checked after every inner sweep. One outer sweep is completed after the set of N_i inner sweeps followed by an adjustment of the local tolerances. The new local tolerance $\varepsilon'_{\text{local}}$ is decreased according to

$$\varepsilon'_{\text{local}} = \varepsilon_{\text{local}} \frac{\varepsilon_V}{V_\psi}, \quad (27)$$

where V_ψ is the actual variance on the current MPS. Equation (27) assumes a linear connection between the local tolerance and the variance fulfilled for small local tolerances [64]. Moreover, we have two more parameters to grow the system up to L sites with the same algorithm later explained

Parameter	Ising model	Long-range Ising model	Bose Hubbard	Spinless Fermi model
χ	46	100	261	82
ε_1	$2.6 \cdot 10^{-9}$	$2.15 \cdot 10^{-8}$	$1.1 \cdot 10^{-10}$	$3.2 \cdot 10^{-10}$
Number of inner sweeps	2	2	2	4
Number of outer sweeps	1	1	1	1
System size L	128	128	32	65
Lanczos iterations	500	500	500	500

Table I. *Empirically Determined Convergence Parameters.* The convergence parameters for the bond dimension χ and Lanczos tolerance ε_1 for four different models achieve a variance tolerance 10^{-12} using a state with high entropy. The Ising model and long-range Ising model are evolved close to the critical point. The Bose-Hubbard model is considered in the superfluid phase and a spinless Fermi model with nearest neighbor repulsive interaction W and nearest neighbor tunneling J is again close to its critical point $J = W$. Details on the study are in Appendix B.

in Sec. IV B for the infinite system. The local tolerance (`warmup_tol`) and the maximal bond dimension (`warmup_bond_dimension`) during that warmup phase can be tuned individually. These values provided in Table I provide a first overview of how models behave within OSMPS. We choose points with high entanglement within each model. The parameters are either close to a critical point or in a phase which has high entanglement such as the superfluid phase of the Bose-Hubbard model.

Finally, we present arguments as to why the variance tolerance is a convenient convergence criterion. In Appendix F we derive the bounds of multiple variables, we provide a short summary of those bounds here. The variance of the ground state V_ψ determines a bound on ϵ , where ϵ is the contribution for $|\psi_\perp\rangle$ of all states orthogonal to the true ground state $|\psi_0\rangle$ in the result $|\psi\rangle$ returned from OSMPS:

$$|\psi\rangle = f |\psi_0\rangle + \epsilon |\psi_\perp\rangle, \quad |\epsilon| \leq \frac{\sqrt{V_\psi}}{\Delta_{0,1}}. \quad (28)$$

The value of $\Delta_{0,1}$ is the energy gap between the ground state and the first excited state. Starting from there, we derive in Appendix F bound on an observable O acting on $|\psi\rangle$ as well as the bond entropy S :

$$|\langle\psi_0| O |\psi_0\rangle - \langle\psi| O |\psi\rangle| \leq 3\epsilon\mathcal{M}, \quad \mathcal{M} = \max_{|\phi\rangle, |\phi'\rangle} |\langle\phi| O |\phi'\rangle|, \quad (29)$$

$$|S(\rho_0) - S(\rho)| \leq \frac{\sqrt{2V_\psi}}{\Delta_{0,1}} \log(D) - \frac{\sqrt{2V_\psi}}{\Delta_{0,1}} \log\left(\frac{\sqrt{2V_\psi}}{\Delta_{0,1}}\right). \quad (30)$$

In Eq. (30) the dimension of the density matrix, D , appears in addition to the variance and the gap.

E. Running the simulations

Finally, we discuss how to set up simulations and execute them on the computer. Each simulation is contained in a dictionary and we can create a list of dictionaries to run multiple simulations at the same time. While certain parameters such as the measurement setup stay the same for a set of simulations, other parameters may be varied. In this example we create an empty list `params`

and add the dictionaries to the list looping over the system size L and the external field h . The dictionary is shown in Listing 5.

Listing 5. Appending different simulations looping over L and h .

```

86     params.append({
87         'simtype'                        : 'Finite',
88         # Filenames and directories
89         'job_ID'                        : 'sim_01_ising_z2_
90         'unique_ID'                    : '_L%04d'%ll + '_h
          %3.6F'%hh,
91         'Write_Directory'              : 'TMP_01_ISING/',
92         'Output_Directory'            : 'OUTPUTS_01_ISING
          /',
93         # System size and Hamiltonian parameters
94         'L'                            : ll,
95         'J'                            : J,
96         'h'                            : hh,
97         # Other parameters
98         'MPSConvergenceParameters'    : conv,
99         'MPSObservables'              : myObservables,
100        # For error calculation (gap)
101        'n_excited_states'              : 1,
102        'eMPSConvergenceParameters'    : conv,
103        'logfile'                      : True,
104        'verbose'                      : 1,
105        # Z2 symmetry
106        'Discrete_generators'           : ['gen'],
107        'Discrete_quantum_numbers'     : [0]
108    })

```

In the following we generate a submit script for our simulations by writing the files for Fortran with a call to

```
MainFiles = mps.WriteMPSParallelFiles(params, Operators, H, hpcsetting,
PostProcess=False)
```

and the simulations are executed when submitted to the computing cluster. The fourth argument `hpcsetting` is a dictionary with various settings such as the number of nodes requested on the cluster. As an alternative, the user may call the parallel executable on a local machine and can find the corresponding call inside the submit script. We do not cover the post-processing itself here, but the sample scripts in Appendix J provide guidance on how to read the results with the corresponding OSMPS functions and access the measurements inside the dictionaries.

IV. HIGHLIGHTS OF STATIC ALGORITHMS

In the previous section we have presented static simulations for the ground state, which builds a basis for other algorithms within OSMPS. The next algorithm searches for the excited state

obtained through variational means. It can find sequentially ascending excited states above the ground state. In addition, we highlight our infinite size statics with an example as a method to calculate properties for the ground state in the thermodynamic limit.

A. Excited state search

The search for excited states, eMPS, is implemented in a successive fashion after the ground state has been obtained. The algorithm is based on the variational procedure now introducing additional Lagrange multipliers to Eq. (24) to enforce orthogonality with previously obtained eigenstates. If the ground state is now labeled as $|\psi_0\rangle$ and the i^{th} excited state as $|\psi_i\rangle$, we then need i additional Lagrange multipliers μ_i corresponding to the number of states with lower energy. These Lagrange multipliers enforce orthogonality between the eigenstates, $\langle\psi_i|\psi_j\rangle = \delta_{i,j}$:

$$\varepsilon[|\psi_i\rangle] = \langle\psi_i|H|\psi\rangle - E\langle\psi_i|\psi_i\rangle - \sum_{j=0}^{i-1} \mu_j \langle\psi_i|\psi_j\rangle. \quad (31)$$

We base our example for the excited state search on the previous study of the quantum Ising model. We introduce long-range interactions following a power law decay in this example. The corresponding Hamiltonian of the long-range Ising model was presented in Eq. (14), and we recall that it was defined as

$$H = -J \sum_{i < j \leq L} \frac{\sigma_i^z \sigma_j^z}{(j-i)^\alpha} - Jh \sum_{i=1}^L \sigma_i^x.$$

The update to the Hamiltonian due to the long-range interaction is reflected in Listing 6. Since we loop over different α , we generate the Hamiltonian as well inside the loop over the different parameters and generate a list of them; in the previous example we could use a single MPO because only the couplings changed, but not the function describing the coupling. For the complete file see Appendix J.

Listing 6. Adapting the MPO for the long-range Ising model.

```

83     H = mps.MPO(Operators, PostProcess=PostProcess)
84     invalpha = lambda x: 1/(x**alpha)
85     H.AddMPOTerm('InfiniteFunction', ['sz', 'sz'],
86                  hparam='J', weight=-1.0, func=invalpha,
87                  L=11, tol=1e-10)
88     H.AddMPOTerm('site', 'sx', hparam='h', weight=-1.0)

```

In order to calculate the excited states, we add the information showed in Listing 7 to the simulation dictionary. In general it is possible to define different observables or convergence parameters for the ground state and the excited state, although it is not possible to have different settings for each excited state. We present the results on the excited states of the long-range Ising model in Fig. 8. The excited states can reveal physical phenomena or support theory, e.g. we deduct from Fig. 8 the close to degenerate ground and first excited state in the ferromagnetic phase. Both the ground and second excited state in the ferromagnetic phase belong to the even sector of the \mathbb{Z}_2 symmetry, and their closing gap indicates the position of the quantum critical

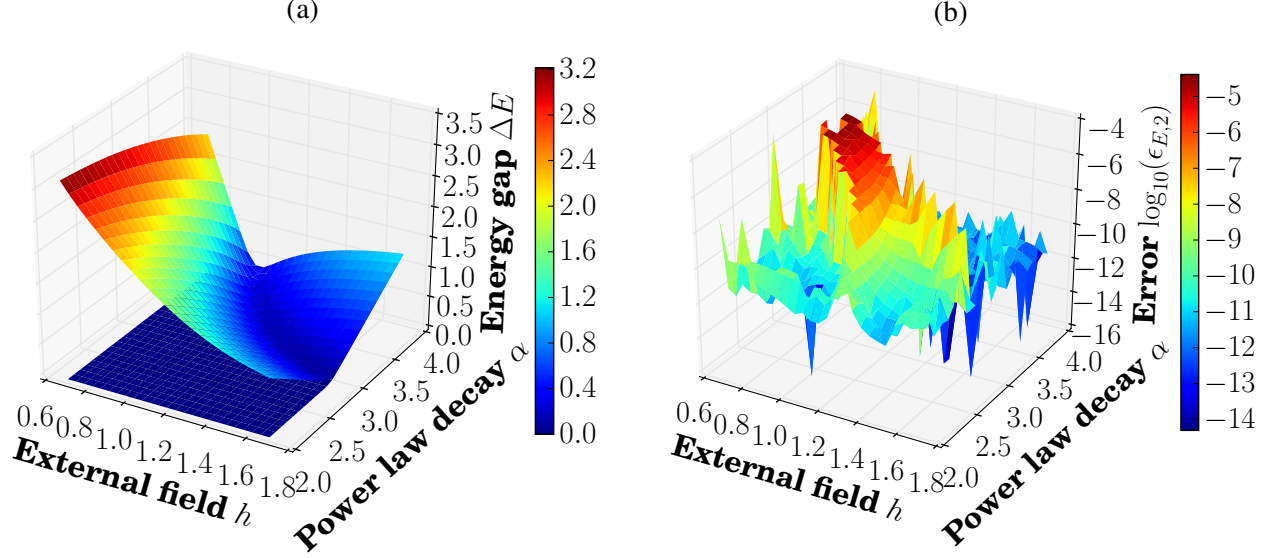


Figure 8. *Energy gap for the long-range quantum Ising model* as a function of the interaction strength α and the external field h . (a) The energy gap between the ground state and the first excited state is close to degeneracy in the ferromagnetic phase of the long-range quantum Ising model. The gap between ground and second excited states closes toward the quantum critical point, e.g. for $\alpha = 4$ around $h \approx 1$. The eigenenergies for the second excited state are compared to the eigenenergies of the ground state and the second excited state of the combined odd and even symmetry sector in (b) to estimate the error. (Same labels apply to color bar and z -axis.)

point. This closing gap can be seen as valley starting around $\alpha = 4$ and $h = 1.0$. We use the symmetry conserving simulation to calculate the ground state and first excited state and show the errors in energy in Fig. 8(b) and (c). In the latter one we see as well the growing error around the critical point. Because the variational state can only guarantee to find an eigenstate, but might end up in a minimum corresponding to a higher excited state, it is useful to resort the energies and converge results with more excited states than actually desired.

Listing 7. Specify the number of excited states to be calculated and the settings for convergence and measurements.

```

110         # eMPS
111         'n_excited_states'           : ne,
112         'eMPSConvergenceParameters' : conv,
113         'eMPSObservables'           : myObservables

```

B. Infinite systems in the thermodynamic limit

The OSMPS library possesses another tool to obtain information about the ground state of a quantum system, which is applicable in the thermodynamic limit. The iMPS algorithm searches for the ground state of a translationally invariant Hamiltonian [56]. The core idea of the algorithm is based on a unit cell of L sites. The Hamiltonian is translationally invariant in the sense that we

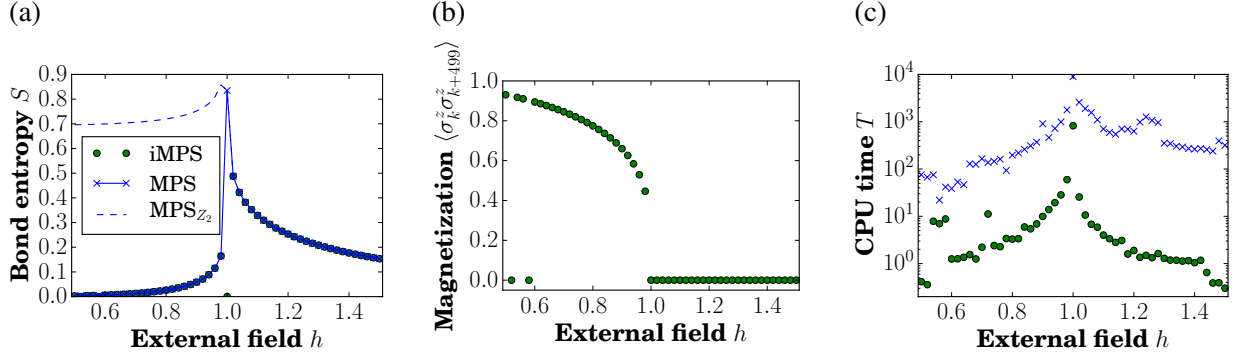


Figure 9. *The quantum Ising model in the infinite MPS simulations.* (a) The bond entropy of the iMPS peaks at the critical point $h_c = 1.0$ which reproduces the results of the finite size MPS simulation for $L = 2000$ without \mathbb{Z}_2 symmetry. In general the bond entropy of the iMPS in the ferromagnetic phase can lie inbetween the results with and without the \mathbb{Z}_2 symmetry. (b) The magnetization based on the correlation $\langle \sigma_i^z \sigma_{i+499}^z \rangle$ dies away at $h = 1.0$. (c) The compute times of the iMPS simulations in comparison with the finite size algorithm at $L = 2000$. iMPS can give a quick estimate of the behavior in the thermodynamic limit.

consider an infinite sequence of these unit cells with the same Hamiltonian. Within the L sites, the Hamiltonian can depend on the site, creating a sublattice or similar features.

The final state is obtained when the state of the unit cell is converged by parameters discussed in the following. Starting with the first unit cell the ground state of the system is obtained. The system size is increased by inserting another unit cell in the middle of the system and summarizing the previous result in an environment. The new ground state of the unit cell is computed under the action of the environment. Subsequent steps of inserting cells while growing the environment lead to the result. The class `iMPSConvParam` comes with the convergence parameters for the bond dimension χ , and the local tolerance and the settings for the Lanczos algorithm keep their meaning in regard to previous algorithms. We introduce the maximal number of iMPS iterations determining how often a new unit cell is introduced into the iMPS state before stopping the algorithm. To break out of the algorithm before reaching the maximal number of iMPS iterations we consider the orthogonality fidelity \mathcal{F} (`variance_tol`). In order to define this orthogonality fidelity \mathcal{F} , we introduce the density matrices ρ_{n-1} , i.e., the density matrix of the previous step, and ρ_n^R as the density matrix of the present step without the new unit cell introduced in step n . The overlap or fidelity serves then as a convergence criterion:

$$\mathcal{F}(\rho_{n-1}, \rho_n^R) = \sqrt{\sqrt{\rho_n^R} \rho_{n-1} \sqrt{\rho_n^R}}. \quad (32)$$

We can use the algorithm to compare the results of the first study of the nearest neighbor limit with those of iMPS. In Fig. 9 we show the bond entropy of the iMPS which peaks at the critical point. We point out that of all simulations, three fail being the second, fifth data point and the bond entropy at the critical point. In comparison we show the largest finite size system with $L = 2000$ with and without using the \mathbb{Z}_2 symmetry. Both lines show good agreement. Possible disagreement in the bond entropy may arise from the actual ground state, i.e. $(|\uparrow \cdots \uparrow\rangle + |\downarrow \cdots \downarrow\rangle) / \sqrt{2}$. But any other superposition of all spins up plus all spins down fulfills the minimization of the energy as well. Furthermore the output for the infinite system is partly different from the finite size algorithm, e.g. the maximal distance for the correlation is specified. More of those differences are described in detail in the manual.

V. TIME EVOLUTION METHODS

The only missing piece to complete the library at this point is the time evolution of quantum states. In total we provide four different algorithms: Krylov time evolution [65] by default, the Time-Dependent Variational Principle algorithm (TDVP) [66], a local Runge-Kutta method (LRK) [67], and a Sornborger-Stewart decomposition [57, 68]. The Sornborger-Stewart decomposition is an alternate decomposition to the Trotter decomposition and is used to implement the Time-Evolving Block Decimation (TEBD) [7]. The first three methods support long-range interactions, and align with our motivation to support such systems.

A. Computational error and convergence

We first provide an overview of each method's behavior in terms of convergence in Fig. 10. The figure compares the OSMPS algorithms to analogous exact diagonalization time propagation schemes, focusing on four key measures:

1. *The maximum trace distance of all local reduced density matrices,*

$$\epsilon_{\text{local}} = \max_{i \in \{1, \dots, L\}} \mathcal{D}(\rho_i, \rho_i^{\text{ED}}), \quad \mathcal{D}(\rho_A, \rho_B) \equiv \frac{1}{2} |\rho_A - \rho_B|, \quad |A| \equiv \sqrt{A^\dagger A}. \quad (33)$$

The superscript *ED* refers to the results of exact diagonalization methods. Equation (33) can be used to bound any local observable, as explained in Appendix G.

2. *The error of correlation measurements.* We consider the maximal trace distance ϵ_{corr} , here on all two-site density matrices, to bound the error for the correlation measurements:

$$\epsilon_{\text{corr}} = \max_{(i,j)} \mathcal{D}(\rho_{i,j}, \rho_{i,j}^{\text{ED}}), \quad i, j \in \{1, \dots, L \mid |i < j|\}. \quad (34)$$

3. *The energy of the system:*

$$\epsilon_E = |E - E^{\text{ED}}|. \quad (35)$$

4. As the maximal bond dimension is one of the key parameters in MPS algorithms, we finally compare the *bond entropy* defined over the von Neumann entropy S of singular values squared obtained by cutting the system in half:

$$\epsilon_S = |S(L/2) - S^{\text{ED}}(L/2)|. \quad (36)$$

The maximal bond dimension necessary in small systems which can still be studied in exact diagonalization is unfortunately limited. Therefore, we cannot study the effects of truncation in comparison to exact diagonalization.

We compare the OSMPS results with the data from exact diagonalization. Therefore, we take the simulation with the smallest time step corresponding to the most accurate result. In addition, we display the error from the static simulation as a lower bound for the error. The static simulation serves as an input state for the dynamics and sets the lower bound for the error. Figure 10 shows the error at the end of the time evolution for a quench in the Ising model. The quench starts at $h(t = 0) = 5.0$ and ends at $h(t = 0.5) = 4.5$ for a system size of $L = 10$. The time is in units of the Ising coupling J . The exact diagonalization method, which is always at time-ordering $\mathcal{O}(dt^2)$, takes the whole Hamiltonian to the exponent evaluating the coupling at $t + 0.5dt$ resulting in $|\psi(T)\rangle = \exp(-iH(0.5dt)dt) \exp(-iH(1.5dt)dt) \cdots \exp(-iH(T - 0.5dt)dt) |\psi(0)\rangle$. In contrast, the MPS time evolution methods support higher order time ordering, which is not used for the studies within this work. We briefly point out the trends within this Fig. 10. We defined the first error as the maximal trace distance over all single site density matrices ϵ_{local} , which is shown in Fig. 10(a). We see two major trends for ϵ_{local} . First, there is a clear difference between the second and fourth order methods in the case of TEBD and LRK algorithms, labeled as TEBD2 and TEBD4 as well as LRK2 and LRK4, respectively. The fourth order algorithms and the TDVP and Krylov algorithms nearly match the result of exact diagonalization for $dt \geq 10^{-3}$ in comparison to the ED result with $dt = 10^{-4}$. Figure 10(b) analyzes the second error, i.e., the error in the reduced two site density matrices ϵ_{corr} . This error is much larger, which is already present in the ground and therefore initial state with an order of magnitude of 10^{-7} . The third kind of error, the error in energy ϵ_E , is shown in Fig. 10(c). ϵ_E decreases for all the methods with the same overall trend. We recall that the Hamiltonian in this case contains single site terms and nearest neighbor terms, so large errors in two site reduced density matrices with sites far apart would not contribute to the error in the energy. The error in the bond entropy ϵ_S , the fourth value considered for the estimate of the error, follows the behavior of the previous measures, see Fig. 10(d). All methods except TEBD2 and LRK2 do not improve from $dt = 10^{-3}$ to $dt = 10^{-4}$ as the entropy is already close to the static result. In general in order to tackle a given problem, we seek for the method which consumes the least resources to achieve a certain error Fig. 10(e) answers this question. We take the error ϵ_{local} as example and plot it as a function of the CPU time. This allows us for example to compare the second order methods versus their fourth order algorithm. Both the TEBD and LRK fourth order methods use less resources at a bigger dt to achieve the same error in comparison to their second order equivalent. The error reported back from OSMPS is analyzed in Fig. 10(f). We consider the Krylov method first: the reported error is bigger for a smaller time step despite the results clearly getting better in the previous plots. The reported Krylov error contains errors from state fitting and cutting the bond dimension. Although it is not necessary to cut the bond dimension, there still remains the possibility for truncation due to the local tolerance criteria. Since the reported error is accumulated during the evolution, the small contributions add up due to the multiple time steps. Considering the error for $dt = 0.0001$ of the order 10^{-9} with 5000 time steps, each time step adds about 10^{-13} to the total error. The other time evolutions report purely the error from truncation of singular values restricted to the local tolerance in this evolution. Increasing errors for decreasing time steps follow the same arguments as for the Krylov time evolution. In Appendix B we discuss in detail the sudden quench and the following evolution under a time-independent Hamiltonian.

The conclusion drawn from this first study are that the total error is bounded by of

$$\epsilon \leq \epsilon_{dt} + \epsilon_{\text{method}} + \epsilon_{\chi}, \quad (37)$$

where ϵ_{dt} is due to evaluating a time-dependent Hamiltonian at discrete points in time. The second error originates from the specific method used to evolve the quantum system in time; an example

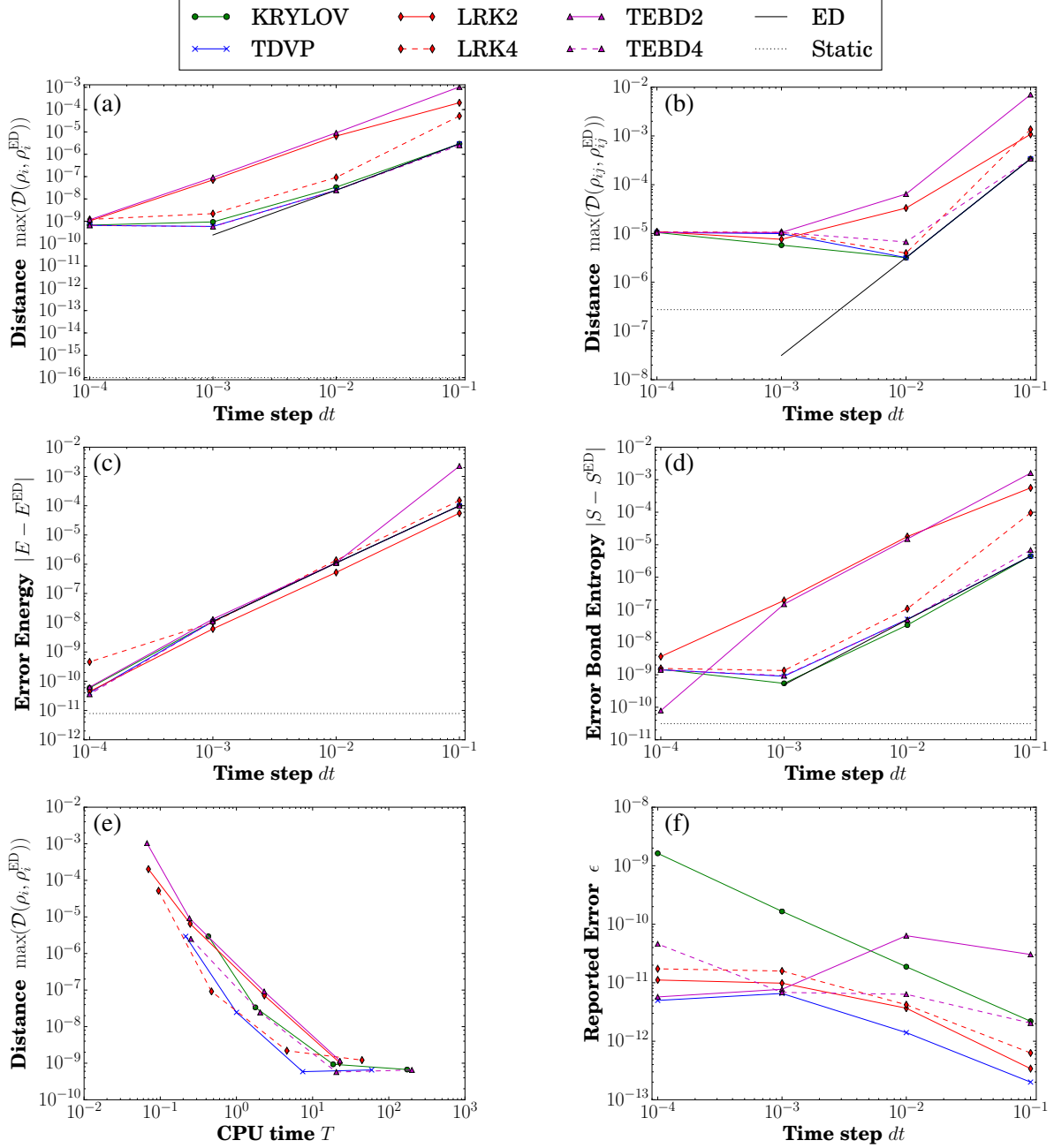


Figure 10. *Scaling of the error in time evolution methods* decreases as expected with the size of the time step. This example shows a quench of the Ising model in the paramagnetic phase. The error decreases as $\mathcal{O}(dt^2)$, the leading order of the error due to time-slicing the time-dependent Hamiltonian $H(t)$ with a CFME. The error for the exact diagonalization method is not plotted for $dt = 10^{-4}$, because this is the result used as a reference and naturally leads to zero error. Curves are a guide to the eye, points represent actual data.

for ϵ_{method} would be the approximation in the Sornborger-Stewart decomposition. Finally, all methods have in common that they truncate singular values to remain with a certain χ leading to ϵ_χ . Equation (37) is an upper bound for the error. A detailed analysis of the error goes beyond

the scope of this work. We emphasize that all three kind of error sources manifest in the four different error measures defined in Eqs. (33) to (36). The scaling of the first error source ϵ_{dt} can be obtained through exact diagonalization since there is neither an error depending on the method nor a truncation of the Hilbert space. ϵ_{dt} appears in the same way for all MPS methods using the same time-ordering as done in this error study. We remain with the estimate of the errors due to the method ϵ_{method} and the truncation in the bond dimension ϵ_{χ} . The Hilbert space for the MPS simulations with $L = 10$ is small enough to capture all singular values and ϵ_{χ} is not present. Therefore, ϵ_{method} can be seen in Fig. 10 if at the same order of magnitude as ϵ_{dt} . For example, the TEBD2 method has an additional error introduced through the Sornborger-Stewart decomposition making it less accurate than the exact diagonalization result with the same time step. We discuss a time-independent Hamiltonian in Appendix B with $\epsilon_{dt} = 0$. Therefore, ϵ_{method} can be estimated independent of the other errors in this case.

Before we discuss particular time-propagation methods, we first discuss how OSMPS accounts for time-ordering of propagators for time-dependent Hamiltonians. When considering time-ordering in MPS algorithms, we want to apply as few operators as possible to avoid increasing the bond dimension, and would like all operators to be easily and efficiently constructed from the MPO form of the Hamiltonian. Therefore, OSMPS uses Commutator-free Magnus expansions (CFMEs) [52, 69]. CFMEs are advantageous over other expressions such as the Dyson series or the original formulation of the Magnus series due to explicit unitarity and the avoidance of nested integrals and/or commutators. OSMPS implements a few different CFMEs with orders of error N , defined such that the propagator is accurate to $\mathcal{O}(\delta t^{N+1})$, and numbers of exponentials s . The default settings are $N = 2$ and $s = 1$, where the CFME amounts to evaluating the Hamiltonian in $[t, t + \delta t]$ using the midpoint rule for integration. Having introduced the general convergence of the methods, we now look at each method individually and discuss their principles.

B. Krylov time evolution

The Krylov method [52, 70, 71] is the default option for real time evolution in the OSMPS code. The main point of using this technique is the support of long-range interactions. The Krylov method applies the exponential of an operator expressed as an MPO to an MPS

$$|\psi(t + dt)\rangle \approx \exp(-iHdt) |\psi(t)\rangle, \quad (38)$$

using the method of Krylov subspace approximations [65, 72]. In the time-independent case H is the Hamiltonian, while in the time-dependent case it is an operator constructed by the particular CFME used.

The Krylov algorithm is not limited to the MPS algorithm, but it is commonly used to obtain the new vector of a matrix exponential acting on a vector. The idea [73] is to change into a truncated basis (the Krylov subspace) $V' = [v_1, v'_2, \dots, v'_n]$ in order to calculate the propagated state $|\psi(t + dt)\rangle = \exp(-iAdt) |\psi(t)\rangle$. The vectors v'_j are chosen as $A^{j-1} |\psi\rangle$ and are orthonormalized to the basis $V = [v_1, v_2, \dots, v_n]$. In particular, the first Krylov vector is $v_1 = |\psi\rangle = V |e_1\rangle$ with $|e_1\rangle = (1, 0, 0, 0, \dots, 0)^T$. Approximating the dot product between the exponential and the state vector leads to

$$\begin{aligned} \exp(-iAdt) |v_1\rangle &\approx VV^\dagger \exp(-iAdt) |v_1\rangle = VV^\dagger \exp(-iAdt) V |e_1\rangle \\ &= V \exp(-iV^\dagger AV dt) |e_1\rangle. \end{aligned} \quad (39)$$

Calculating the exponential M of the matrix $V^\dagger AV \in \mathbb{R}^{n \times n}$ is a numerically feasible task as long as the number of basis vectors n is much smaller than the dimension of the Hilbert space D .

Furthermore, the relation simplifies to a real tridiagonal matrix for Hermitian matrices, which is satisfied by the Hamiltonian. This leads to

$$\exp(-iAdt) |v_1\rangle \approx VM |e_1\rangle = \sum_{i=1}^n M_{1,i} |v_i\rangle . \quad (40)$$

While in many applications where the state is represented as a vector this is enough to obtain the approximation of $|\psi(t+dt)\rangle$, the problem in the case of the MPS is that the summation over $|v_i\rangle$ can not be exactly carried out, as the set of MPSs with fixed bond dimension do not form a vector space. Based on the previous approaches using variational algorithms, we instead find $|\psi(t+dt)\rangle$ by variationally optimizing the overlap

$$\langle\psi(t+dt)| \left(\sum_{i=1}^n m_{1,i} |v_i\rangle \right) . \quad (41)$$

This procedure is done optimizing local tensors as in the ground state search. However, instead of solving an eigenvalue problem at each iteration, this optimization takes the form of a linear system of equations. By exploiting the isometrization of MPSs (see Eq. (7)), this linear system of equations can be made into an equality. The interested reader can find further details on this algorithm in Refs. [8, 52].

C. Sornborger-Stewart decomposition

This implementation inside the OSMPS library is suitable for nearest-neighbor Hamiltonians. The Sornborger-Stewart decomposition [68] used in the OSMPS algorithms sweeps through the system acting on every site, instead of every second pair of sites as in a more common alternative Suzuki-Trotter decomposition [57]. The second order expansion takes the form

$$\exp\left(-idt \sum_{i=1}^{L-1} H_{i,i+1}\right) = \prod_{i=1}^{L-1} \exp\left(-i\frac{dt}{2} H_{i,i+1}\right) \prod_{i=1}^{L-1} \exp\left(-i\frac{dt}{2} H_{L-i,L-i+1}\right) . \quad (42)$$

We again follow the Krylov approach to propagate the quantum state under the given MPO taking the exponential in the Krylov subspace. The essential difference is the local characteristics of the Hamiltonian in the Sornborger-Stewart decomposition. With the orthogonality center at one of the sites i and $j = i + 1$ being acted on, the overlap from the left and right is the identity operator. If we denote the two sites acted on with $|C\rangle$ and the parts to the left of i and right of j with $|L_i\rangle$ and $|R_j\rangle$, the actual state vector $|\psi\rangle$ can be derived from Eq. (40),

$$|\psi\rangle = \sum_{i,j} |L_i\rangle |C_{i,j}\rangle |R_j\rangle . \quad (43)$$

Within the construction of the Krylov basis $[v_1, \dots, v_n]$ the states $|L_i\rangle$ and $|R_j\rangle$ remain unchanged as shown in Appendix H. The same applies for the sum of the propagated state

$$|\psi(t+dt)\rangle = \sum_k c_k v_k , \quad (44)$$

leading to the following construction of the state in the MPS picture:

$$|\psi(t + dt)\rangle = \sum_{i,j} |A_i\rangle \left(\sum_k c_k |C_{i,j}(v_k)\rangle \right) |B_j\rangle . \quad (45)$$

That means that we can sum locally over the two site tensors, as they form a vector space, in contrast to the long-range case where we had to variationally find the MPS closest to the summation.

In order to specify the error due to the method, we have to consider the decomposition of the exponential. By separating non-commuting terms in matrix exponential, we get an error of order dt^2 in the first order approximation for a single time step:

$$\exp[(A + B)dt] = \exp(Adt) \exp(Bdt) + \mathcal{O}(dt^2) . \quad (46)$$

Having implemented the second and fourth order approximations we obtain methodical errors ϵ_{method} for the whole time evolution as follows. ϵ_{method} is defined as part of the total error in Eq. (37).

$$\epsilon_{\text{TEBD2}} = \mathcal{O}(dt^2) , \quad \epsilon_{\text{TEBD4}} = \mathcal{O}(dt^4) . \quad (47)$$

In addition to the error of the Sornborger-Stewart decomposition, we have as well an error from the Krylov subspace approximation to the exponential for the local two-site propagators. The error bound for a single step is derived in [72].

The convergence parameters necessary to set up time evolution with TEBD methods reflect the simplicity of the approach. The parameter `psi_local_tol` determines the local truncation on the singular values, while the pair `(lanczos_tol, max_num_lanczos_iter)` provides the tolerance for the Krylov approximation and the maximal number of Krylov vectors. In addition, the maximum bond dimension χ can be defined.

With regards to the convergence study for the quench in the Ising model in Fig. 10 we make two observations. The fourth order Sornborger-Stewart method is better than second order implementation of Sornborger-Stewart, as expected due to the smaller error at each time step. The rate of convergence as a function of the time step dt , that is, the slope of the line, is equal for both implementations. The error ϵ_{dt} from the time-ordering of the time-dependent Hamiltonian governs both implementations with $\mathcal{O}(dt^2)$ and the better convergence of the fourth order Sornborger-Stewart cannot be observed. In contrast, if the Hamiltonian is time-independent, there is no error from the time-ordering. Figure. 15 in Appendix B indicates that TEBD4 has a higher rate of convergence in this case. The total error has to be considered in comparison to the Krylov time evolution in Sec. VB or the TDVP discussed in Sec. VD. We remark that the fourth order TEBD method has a comparable error to the Krylov and TDVP method within one order of magnitude for the smallest time step $dt = 10^{-4}$. Larger time steps and the second order method TEBD2 introduce errors which are sometimes two orders of magnitude larger than the errors introduced through Krylov or TDVP, especially for large time steps.

D. Time-dependent variational principle

As a third option for the time evolution of a quantum system, we provide the time-dependent variational principle (TDVP) [66]. This is another method supporting long-range interactions. In brief, all other previous methods apply the propagator, which is an entangling many-body operator, to a state represented as an MPS, and produces a new state obtained as an MPS. The updated

MPS has a larger bond dimension in general, so then we must variationally project this new MPS onto the set of states with reduced computational resources. The approach of the TDVP method is instead to project the time-dependent Schrödinger equation onto the manifold of MPSs with fixed bond dimension, and then integrate this equation directly within this manifold. OSMPS has implemented the two-site version of this algorithm [66], in which the bond dimension is still allowed to grow over the course of time evolution, but the propagator is determined from a projection of the full many-body operator onto a more local subspace. We remark that TDVP performs well on the convergence study against exact diagonalization methods in the example of Fig. 10. All four estimates for the error defined in the Eqs. (33) to (36) and shown in the four upper panels of Fig. 10 are close to exact diagonalization result with the same time step used as reference. The maximal distance of all reduced two-site density matrices does not reach the reference due to initial errors in the static results for the ground state.

E. Local Runge-Kutta propagation

Another option for time evolution with long-range interactions available in OSMPS is the local Runge-Kutta method proposed in [67]. The basic idea can be summarized as using the Runge-Kutta method on the local MPO matrices instead of the whole propagator. The new MPO representing the propagator U_{LRK} has a compact representation, i.e., it does not increase in bond dimension beyond that of the Hamiltonian MPO. Since the method is defined on the MPO, it is the third method supporting long-range interactions.

A detailed overview on how to build the MPO for the propagator, U_{LRK} , as a second order approximation is beyond the scope of this description; we suggest [67] to the interested reader. But we do provide here a short description of how the application of the MPO to the MPS is implemented in OSMPS. We fit the product of the MPO and the MPS to a new MPS, where the new MPS represents the propagated state,

$$\min_{|\psi(t+dt)\rangle} \| |\psi(t+dt)\rangle - U_{\text{LRK}} |\psi(t)\rangle \| . \quad (48)$$

This fit employs the same method used to fit the next Krylov vector $|v_{i+1}\rangle = H |v_i\rangle$ in the Krylov method. The four upper panels of Fig. 10 shows a similar behavior with regards to the convergence of the LRK method as for the TEBD method. The higher order variant of the LRK method reduces the error, but the rate of convergence is not better due to the time-dependent Hamiltonian and the particular choice of CFME. Within the implementation of this time evolution method we use the EXPOKIT package [74, 75] to calculate matrix exponentials.

F. Time evolution case study: Bose-Hubbard model in a rotating saddle point potential

We consider bosons in an optical lattice confined in a rotating potential as an example of to set up time evolution in OSMPS. We consider the potential V_{xy} with a saddle point at $x = y = 0$ and a weight c ,

$$V_{xy} = c (x^2 - y^2) . \quad (49)$$

Figure 11(a) shows this potential for a two dimensional lattice. We consider a one-dimensional optical lattice in this potential marked by the red sites and start to rotate V_{xy} ; the one-dimensional

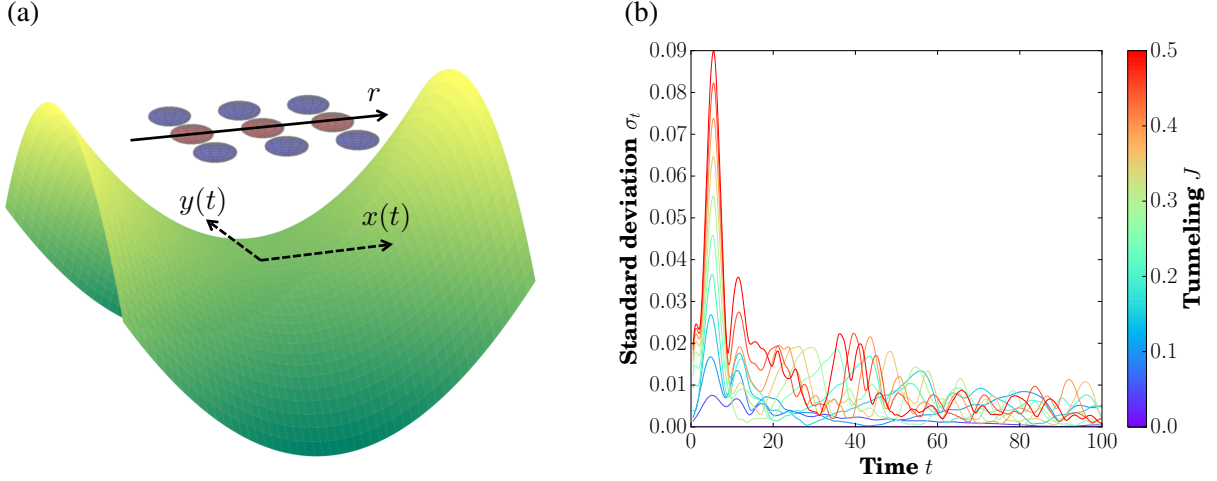


Figure 11. *Bose-Hubbard model in a rotating saddle-point potential.* The systems shows an increasing stability for faster frequencies. (a) In order to simulate a one dimensional system from the two dimensional optical lattice, we consider one slice of sites marked in red with its coordinate system r . The saddle point potential in green and yellow rotates under the system characterized by the coordinates $x(t), y(t)$. (b) The standard deviation over time intervals of $\tau = 5$ is measured for the spreading in x , i.e. the standard deviation of the number operator. This observable can be expressed as $\sigma_t(\sigma_x(n), \tau = 5)$.

system sees the following potential V_S depending on the angle of rotation ϕ and the distance r from the center of the potential and integrate it into the Bose-Hubbard Hamiltonian:

$$V_S = c (r \cos(\phi(t)))^2 - c (r \sin(\phi(t)))^2, \quad (50)$$

$$H = -J \sum_{i=1}^{L-1} b_i b_{i+1} + h.c. + \frac{U}{2} \sum_{i=1}^L n_i (n_i - \mathbb{I}) + \sum_{i=1}^L V_S(t, i) n_i. \quad (51)$$

We slowly ramp up the frequency of the rotations with an acceleration α starting with a flat potential

$$\phi(t) = \frac{\pi}{4} + \frac{1}{2} \alpha t^2. \quad (52)$$

In order to estimate the stability of the system, we first calculate the standard deviation of the number operator with regards to spatial dimension x for every measurement in time, denoted by $\sigma_x(n)$ ². We calculate $\sigma_t(\sigma_x(n), \tau = 5)$, that is the standard deviation of $\sigma_x(n)$ for time intervals $t - \tau$ to t for $\tau = 5$, in a second step. For fast enough frequencies the rotating potential stabilizes the system, which can be seen in Fig. 11(b). The other trend is that towards the superfluid regime, with higher tunneling, the stability decreases. The first peak shows this trend.

The setup of the dynamics has many common steps with the statics. The definition of the operators, the MPO representing the Hamiltonian, and the observable class do not change. In regard to the time-dependent potential V_S in Eq. (50), we define the weight c , the acceleration α (alpha), the system size L , and the function returning the corresponding potential at a point in time t . The latter step is shown in Listing 8.

² We emphasize that this is not the Pauli operator, but the standard derivation using in contrast to the Pauli operators the subscript

Listing 8. Define a time-dependent function for the rotating saddle potential.

```

68  def Vt(t, c=c, alpha=alpha, L=L):
69      # Grid with particles symmetric around 0 at unit distance
70      grid = np.linspace(-L / 2 + 0.5, L / 2 - 0.5, L)
71
72      phit = np.pi / 4 + 0.5 * alpha * t**2
73
74      return c * ((grid * np.cos(phit))**2 - (grid * np.sin(phit)
              )**2)

```

Next, we build the time evolution. As before with MPOs, observables, and convergence parameters, the dynamics are contained in an instance of a Python class: `QuenchList`. Once created, we can add multiple quenches which are executed sequentially. We only add one quench in our example in Listing 9. Different convergence parameters are supported for each quench you add.

Listing 9. Creating the class object containing one or more quenches.

```

76  tconv = mps.TEBDCnvParam(max_bond_dimension=60)
77  Quench = mps.QuenchList(H)
78  Quench.AddQuench(['Vt'], 100.0, 0.01, [Vt], stepsforoutput=10,
79                  ConvergenceParameters=tconv)

```

As a last step we have to specify the dynamics in the dictionary for the simulation, including the quenches and the observables to be measured during the dynamics. The additional lines are shown in Listing 10. The initial state of the time evolution is the ground state if not specified otherwise.

Listing 10. Additional dictionary entries for the time evolution.

```

102  'Quenches' : Quench,
103  'DynamicsObservables' : myObs,

```

The post processing works similar to the statics. A listed list contains the dictionaries with the results of the simulation. The outer list contains the different simulations, and the inner list contains the measurements for each time. The complete Python code may be found in Appendix J.

VI. FUTURE DEVELOPMENTS

The present OSMPS library has a wide field of possible applications including two-component mixtures [41], topological phases [46], and complexity in the quantum mutual information [51]. Still, we are planing to enhance the code to solve other sets of problems. The need for enhancements is driven by our research in atomic, molecular, and optical physics, macroscopic quantum phenomena, and complexity, but we are open to suggestions from the community for extensions to the OSMPS library. For instance, the feature to provide the singular values of a bipartition in addition to its bond entropy began as suggestions made on our developer's site [76]. We already have a broad suite of tools for the current MPS algorithms for finite systems; therefore, we are focusing on implementing additional measurements or new MPO rule sets to target Hamiltonians which are

impossible or difficult to build with present rule sets. For the measurements, observables such as unequal time correlations are a possible add-on. A user-friendly support for ladder systems or rectangular systems with $L_x \gg L_y$ could be one focus for new types of Hamiltonians, either as new rule sets or with convenient interfaces in Python. In contrast, other tensor network structures such as PEPS are not being considered as an extension to the library at this point. Periodic boundary conditions are on the agenda for selected MPO rule sets.

On the other hand, we intend to extend OSMPS for a new set of problems in the future, namely open quantum systems. We anticipate them to be key component to establish a more realistic picture of simulations with regards to experiments, for example, future quantum computers will suffer effects such as decoherence from the coupling to the environment; including those effects in simulations fosters the understanding and addresses problems induced by open systems. The standard approach is the Lindblad master equation describing a system coupled weakly to a Markovian environment. Several approaches have been proposed to implement the Lindblad master equation, i.e. quantum trajectories, MPDOs, and Locally Purified Tensor Networks (LPTNs) [77]. We are developing as well techniques to evolve open quantum systems with a non-Markovian environment within the tensor network algorithms.

Finally, small features are on the list of future developments, e.g. providing optional support for results in HDF5 file format for more convenient data post processing, improving the speedup when using shared-memory parallelization with openMP, and a TEBD algorithm which is not based on the Krylov approximation.

VII. CONCLUSIONS

In this paper we have presented a description of the MPS library OSMPS containing a set of powerful tools to study static and dynamic properties of entangled one-dimensional many body quantum systems, where the initial focus is on a broad set of methods for quantum simulators based on atomic, molecular and optical physics architectures. These algorithms and interfaces are widely generalizable to other fields of quantum physics, e.g. condensed matter and materials modeling. The usefulness of our methods is underscored by the rapid community adaption and subsequent publications in various areas of quantum physics making use of the OSMPS library or a derivative based on OSMPS [40–55]. OSMPS has been downloaded more than 2300 times from over 55 countries since its initial release on SourceForge in January 2014.

We combine a Fortran core with an easy to use Python front end. By introducing the Python interface we hide the complex structures of the core algorithms from the end user. Moreover, Python is a simpler programming language lowering the barrier to start simulations for non-specialists and students. OSMPS provides fast and easy-to-access numerical predictions for 1D quantum many-body systems. We facilitate this learning process by the recent integration of a small exact diagonalization package inside OSMPS written purely in Python [78]. On the other hand, we also support sophisticated enough features to provide helpful tools for quantum many-body theorists and specialists in quantum computational physics. We presented these tools throughout the paper describing the features of the library, which include ground states of infinite systems and ground states plus low lying excited states in finite size systems. For the dynamics of finite size systems, we provided four different tools starting from a nearest-neighbor Sornborger-Stewart decomposition, a modified Trotter approach, to methods supporting long-range interactions such as Krylov subspaces, local Runge-Kutta, and the time dependent variational principle. We gave a detailed description of the nuanced convergence properties of these methods.

We anticipate that many unexplored problems in quantum simulators, materials modeling, and

other areas are suitable for OSMPS, starting with long-range quantum physics and its still unexplored corners reaching to less studied models, e.g. facets of the XYZ model, and the vast variety of untouched far from equilibrium dynamics. The development of new features in OSMPS, as described in Sec. VI, follows new emerging fields in quantum physics which come naturally with explorations and requests by the community of both end users and developers.

Providing the library as an open source package including a dedicated forum fosters continued community development and research in many-body entangled quantum physics with a transparent tool, much as density function theory (DFT) was instrumental to the materials genome initiative. Especially the aspect of modifying the core code to integrate tailored tools on top or integrate modules into other open source packages strengthens the idea of cooperative research. The version 2.0 of our library described in this paper is available under the *GNU General Public License* (GPL3) [79] on our homepage <http://sourceforge.net/projects/openmps/>.

ACKNOWLEDGMENTS

We gratefully appreciate contributions from and discussions with A. Dhar, B. Gardas, A. Glick, W. Han, D. M. Larue, and D. Vargas during the development of OSMPS. We are equally thankful to the ALPS collaboration [25, 80] and to C. W. Clark, J. E. Williams, I. Danshita, B. I. Schneider, and R. Mishmash who contributed heavily to the predecessor of OSMPS, OpenTEBD [33]. The calculations were carried out using the high performance computing resources provided by the Golden Energy Computing Organization at the Colorado School of Mines. This work has been supported by the NSF under the grants PHY-120881 and PHY-1520915 and the AFOSR under grant FA9550-14-1-0287.

-
- [1] Steven R. White, “Density matrix formulation for quantum renormalization groups,” *Phys. Rev. Lett.* **69**, 2863–2866 (1992).
 - [2] Steven R. White, “Density-matrix algorithms for quantum renormalization groups,” *Phys. Rev. B* **48**, 10345–10356 (1993).
 - [3] N. V. Prokof’ev, B. V. Svistunov, and I. S. Tupitsyn, “Exact, complete, and universal continuous-time worldline Monte Carlo approach to the statistics of discrete quantum systems,” *Journal of Experimental and Theoretical Physics* **87**, 310–321 (1998).
 - [4] Anders W. Sandvik and Juhani Kurkijärvi, “Quantum Monte Carlo simulation method for spin systems,” *Phys. Rev. B* **43**, 5950–5961 (1991).
 - [5] Anatoli Polkovnikov, “Phase space representation of quantum dynamics,” *Annals of Physics* **325**, 1790–1852 (2010).
 - [6] J. Schachenmayer, A. Pikovski, and A. M. Rey, “Many-Body Quantum Spin Dynamics with Monte Carlo Trajectories on a Discrete Phase Space,” *Phys. Rev. X* **5**, 011022 (2015).
 - [7] Guifré Vidal, “Efficient classical simulation of slightly entangled quantum computations,” *Phys. Rev. Lett.* **91**, 147902 (2003).
 - [8] Ulrich Schollwöck, “The density-matrix renormalization group in the age of matrix product states,” *Annals of Physics* **326**, 96 – 192 (2011), january 2011 Special Issue.
 - [9] Román Orús, “A practical introduction to tensor networks: Matrix product states and projected entangled pair states,” *Annals of Physics* **349**, 117 – 158 (2014).

- [10] G. Kin-Lic Chan, A. Keselman, N. Nakatani, Z. Li, and S. R. White, “Matrix Product Operators, Matrix Product States, and ab initio Density Matrix Renormalization Group algorithms,” [ArXiv e-prints 1605.02611](#) (2016).
- [11] Sukhwinder Singh, Robert N. C. Pfeifer, and Guifré Vidal, “Tensor network decompositions in the presence of a global symmetry,” [Phys. Rev. A](#) **82**, 050301 (2010).
- [12] Sukhwinder Singh, Robert N. C. Pfeifer, and Guifré Vidal, “Tensor network states and algorithms in the presence of a global U(1) symmetry,” [Phys. Rev. B](#) **83**, 115125 (2011).
- [13] F. Verstraete, V. Murg, and J.I. Cirac, “Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems,” [Advances in Physics](#) **57**, 143–224 (2008).
- [14] Y.-Y. Shi, L.-M. Duan, and G. Vidal, “Classical simulation of quantum many-body systems with a tree tensor network,” [Phys. Rev. A](#) **74**, 022320 (2006).
- [15] Jean Dalibard, Yvan Castin, and Klaus Mølmer, “Wave-function approach to dissipative processes in quantum optics,” [Phys. Rev. Lett.](#) **68**, 580–583 (1992).
- [16] R. Dum, P. Zoller, and H. Ritsch, “Monte carlo simulation of the atomic master equation for spontaneous emission,” [Phys. Rev. A](#) **45**, 4879–4887 (1992).
- [17] F. Verstraete, J. García-Ripoll, and J. Cirac, “Matrix Product Density Operators: Simulation of Finite-Temperature and Dissipative Systems,” [Phys. Rev. Lett.](#) **93**, 207204 (2004).
- [18] Michael Zwolak and Guifré Vidal, “Mixed-state dynamics in one-dimensional quantum lattice systems: A time-dependent superoperator renormalization algorithm,” [Phys. Rev. Lett.](#) **93**, 207205 (2004).
- [19] L. D. Carr, *Understanding Quantum Phase Transitions*, Condensed Matter Physics (CRC Press, Boca Raton, FL, 2010).
- [20] G. Vidal, “Class of Quantum Many-Body States That Can Be Efficiently Simulated,” [Phys. Rev. Lett.](#) **101**, 110501 (2008).
- [21] E. M. Stoudenmire and Steven R. White, “Minimally entangled typical thermal state algorithms,” [New Journal of Physics](#) **12**, 055026 (2010).
- [22] Antoine Georges, Gabriel Kotliar, Werner Krauth, and Marcelo J. Rozenberg, “Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions,” [Rev. Mod. Phys.](#) **68**, 13–125 (1996).
- [23] Erich Runge and E. K. U. Gross, “Density-functional theory for time-dependent systems,” [Phys. Rev. Lett.](#) **52**, 997–1000 (1984).
- [24] This algorithm is not appropriate for capturing entanglement dynamics and strong correlations.
- [25] “Algorithms and Libraries for Physics Simulations (ALPS),” <http://alps.comp-phys.org>, last visited Feb 27, 2017.
- [26] “BLOCK – DMRG for quantum chemistry,” <http://sanshar.github.io/Block/>, last visited Feb 27, 2017.
- [27] Sebastian Wouters, Ward Poelmans, Paul W. Ayers, and Dimitri Van Neck, “CheMPS2: A free open-source spin-adapted implementation of the density matrix renormalization group for ab initio quantum chemistry,” [Computer Physics Communications](#) **185**, 1501 – 1514 (2014).
- [28] “DMRG++,” <https://web.ornl.gov/gz1/dmrgPlusPlus/index.html>, last visited Feb 27, 2017.
- [29] J. García-Ripoll, “Matrix product states,” <http://github.com/juanjosegarciaripoll/mps>, last visited Feb 27, 2017.
- [30] “ITensor – Intelligent Tensor,” <http://itensor.org/>, last visited Feb 27, 2017.
- [31] Piet Dargel and Thomas Köhler, “MPS-DMRG Applet,” <http://www.theorie.physik.uni-goettingen.de/thomas.koehler/doku.php?id=en:start>, last visited Feb 27, 2017.
- [32] Ashley Milsted, “evoMPS,” <http://github.com/amilsted/evoMPS>, last visited Feb 27, 2017.

- [33] “OpenTEBD: Open Source Time-Evolving Block Decimation,” <http://sourceforge.net/projects/opentebd/>, last visited Feb 27, 2017.
- [34] James R. Garrison and Ryan V. Mishmash, “Simple DMRG,” <http://github.com/simple-dmrg/simple-dmrg/>, last visited Feb 27, 2017.
- [35] “Snake DMRG,” <http://github.com/entron/snake-dmrg>, last visited Feb 27, 2017.
- [36] R. Fazio, S. Montangero, G. De Chiara, D. Rossini, M Rizzi, and D. Bleh, “Powder with Power DMRG,” <http://qti.sns.it/dmrg/phome.html>, last visited Feb 27, 2017.
- [37] Miroslav Urbanek and Pavel Soldán, “Parallel implementation of the time-evolving block decimation algorithm for the Bose-Hubbard model,” *Computer Physics Communications* **199**, 170 – 177 (2016).
- [38] “Uni10 – Universal Tensor Network Library,” <http://yingjerkao.github.io/uni10/>, last visited Feb 27, 2017.
- [39] “Open Source Matrix Product States (OpenMPS),” <http://sourceforge.net/projects/openmps/>, last visited Feb 27, 2017.
- [40] E. Anisimovas, M. Račiūnas, C. Sträter, A. Eckardt, I. B. Spielman, and G. Juzeliūnas, “Semisynthetic zigzag optical lattice for ultracold bosons,” *Phys. Rev. A* **94**, 063632 (2016).
- [41] Filipe F. Bellotti, Amin S. Dehkharghani, and Nikolaj T. Zinner, “Comparing numerical and analytical approaches to strongly interacting two-component mixtures in one dimensional traps,” *The European Physical Journal D* **71**, 37 (2017).
- [42] A. Dhar, J. J. Kinnunen, and P. Törmä, “Population imbalance in the extended Fermi-Hubbard model,” *Phys. Rev. B* **94**, 075116 (2016).
- [43] Michele Dolfi, Bela Bauer, Sebastian Keller, Alexandr Kosenkov, Timothée Ewart, Adrian Kantian, Thierry Giamarchi, and Matthias Troyer, “Matrix product state applications for the ALPS project,” *Computer Physics Communications* **185**, 3430 – 3440 (2014).
- [44] B. Gardas, J. Dziarmaga, and W. H. Zurek, “Quench in the 1D Bose-Hubbard model,” *ArXiv e-prints* 1612.05084 (2016).
- [45] Z.-X. Gong, M. F. Maghrebi, A. Hu, M. Foss-Feig, P. Richerme, C. Monroe, and A. V. Gorshkov, “Kaleidoscope of quantum phases in a long-range interacting spin-1 chain,” *Phys. Rev. B* **93**, 205115 (2016).
- [46] Z.-X. Gong, M. F. Maghrebi, A. Hu, M. L. Wall, M. Foss-Feig, and A. V. Gorshkov, “Topological phases with long-range interactions,” *Phys. Rev. B* **93**, 041102 (2016).
- [47] D. Jaschke, K. Maeda, J. D. Whalen, M. L. Wall, and L. D. Carr, “Critical Phenomena and Kibble-Zurek Scaling in the Long-Range Quantum Ising Chain,” *ArXiv e-prints* 1612.07437 (2016).
- [48] Andrew P. Koller, Michael L. Wall, Josh Mundinger, and Ana Maria Rey, “Dynamics of Interacting Fermions in Spin-Dependent Potentials,” *Phys. Rev. Lett.* **117**, 195302 (2016).
- [49] M. F. Maghrebi, Z.-X. Gong, and A. V. Gorshkov, “Continuous symmetry breaking and a new universality class in 1D long-range interacting quantum systems,” *ArXiv e-prints* 1510.01325 (2015).
- [50] Angelo Russomanno and Emanuele G. Dalla Torre, “Kibble-Zurek scaling in periodically driven quantum systems,” *Europhysics Letters* **115**, 30006 (2016).
- [51] D. L. Vargas and L. D. Carr, “Detecting Quantum Phase Transitions via Mutual Information Complex Networks,” *ArXiv e-prints* 1508.07041 (2015).
- [52] M. L. Wall and Lincoln D. Carr, “Out-of-equilibrium dynamics with matrix product states,” *New Journal of Physics* **14**, 125015 (2012).
- [53] M. L. Wall, Erman Bekaroglu, and Lincoln D. Carr, “Molecular Hubbard Hamiltonian: Field regimes and molecular species,” *Phys. Rev. A* **88**, 023605 (2013).
- [54] M. L. Wall and L. D. Carr, “Dipole-dipole interactions in optical lattices do not follow an inverse cube power law,” *New Journal of Physics* **15**, 123005 (2013).

- [55] Hendrik Weimer, “String order in dipole-blockaded quantum liquids,” *New Journal of Physics* **16**, 093040 (2014).
- [56] I. P. McCulloch, “Infinite size density matrix renormalization group, revisited,” *ArXiv e-prints* 0804.2509 (2008).
- [57] Michael A. Nielsen and Isaac L. Chuang, *Quantum Computation and Quantum Information*, 9th ed. (Cambridge Univ. Press, Cambridge, United Kingdom, 2007).
- [58] J. Eisert, M. Cramer, and M. B. Plenio, “Colloquium : Area laws for the entanglement entropy,” *Rev. Mod. Phys.* **82**, 277–306 (2010).
- [59] F. Verstraete, D. Porras, and J. I. Cirac, “Density Matrix Renormalization Group and Periodic Boundary Conditions: A Quantum Information Perspective,” *Phys. Rev. Lett.* **93**, 227205 (2004).
- [60] Ernst Ising, “Beitrag zur Theorie des Ferromagnetismus,” *Zeitschrift für Physik* **31**, 253–258 (1925).
- [61] Amit Dutta and J. K. Bhattacharjee, “Phase transitions in the quantum Ising and rotor models with a long-range interaction,” *Phys. Rev. B* **64**, 184106 (2001).
- [62] Gregory M. Crosswhite and Dave Bacon, “Finite automata for caching in matrix product algorithms,” *Phys. Rev. A* **78**, 012356 (2008).
- [63] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins Studies in Mathematical Sciences (The Johns Hopkins University Press, Baltimore, MD, 1996).
- [64] Ian P. McCulloch, “From density-matrix renormalization group to matrix product states,” *Journal of Statistical Mechanics: Theory and Experiment* **2007**, P10014 (2007).
- [65] Y. Saad, “Analysis of Some Krylov Subspace Approximations to the Matrix Exponential Operator,” *SIAM Journal on Numerical Analysis* **29**, 209–228 (1992).
- [66] Jutho Haegeman, Christian Lubich, Ivan Oseledets, Bart Vandereycken, and Frank Verstraete, “Unifying time evolution and optimization with matrix product states,” *Phys. Rev. B* **94**, 165116 (2016).
- [67] Michael P. Zaletel, Roger S. K. Mong, Christoph Karrasch, Joel E. Moore, and Frank Pollmann, “Time-evolving a matrix product state with long-ranged interactions,” *Phys. Rev. B* **91**, 165112 (2015).
- [68] A. T. Sornborger and E. D. Stewart, “Higher-order methods for simulations on quantum computers,” *Phys. Rev. A* **60**, 1956–1965 (1999).
- [69] A. Alvermann and H. Fehske, “High-order commutator-free exponential time-propagation of driven quantum systems,” *Journal of Computational Physics* **230**, 5930 – 5956 (2011).
- [70] Salvatore R. Manmana, Alejandro Muramatsu, and Reinhard M. Noack, “Time evolution of one-dimensional quantum many body systems,” *AIP Conference Proceedings* **789**, 269–278 (2005).
- [71] Juan José García-Ripoll, “Time evolution of matrix product states,” *New Journal of Physics* **8**, 305 (2006).
- [72] E. Gallopoulos and Y. Saad, “Efficient Solution of Parabolic Equations by Krylov Approximation Methods,” *SIAM Journal on Scientific and Statistical Computing* **13**, 1236–1264 (1992), <http://dx.doi.org/10.1137/0913071>.
- [73] Cleve Moler and Charles Van Loan, “Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later,” *SIAM Review* **45**, 3–49 (2003).
- [74] “EXPOKIT,” <http://www.maths.uq.edu.au/expokit/>, last visited Feb 27, 2017.
- [75] R. B. Sidje, “EXPOKIT. A Software Package for Computing Matrix Exponentials,” *ACM Trans. Math. Softw.* **24**, 130–156 (1998).
- [76] The request were made in the questions on [fermionic systems](#) and [MPS simulations](#).
- [77] A. H. Werner, D. Jaschke, P. Silvi, M. Kliesch, T. Calarco, J. Eisert, and S. Montangero, “Positive tensor network approach for simulating open quantum many-body systems,” *Phys. Rev. Lett.* **116**, 237201 (2016).

- [78] D. Jaschke and L. D. Carr, “Open source Matrix Product States: Exact diagonalization and other entanglement-accurate methods revisited in quantum systems,” in prep. (2017).
- [79] “GNU General Public License,” <http://www.gnu.org/licenses/gpl-3.0.en.html>, last visited Feb 27, 2017.
- [80] B. Bauer, L. D. Carr, H. G. Evertz, A. Feiguin, J. Freire, S. Fuchs, L. L. Gamper, J. Gukelberger, E. Gull, S. Guertler, A. Hehn, R. Igarashi, S. V. Isakov, D. Koop, P. N. Ma, P. Mates, H. Matsuo, O. Parcollet, G. Pawłowski, J. D. Picon, L. Pollet, E. Santos, V. M. Scarola, U. Schollwöck, C. Silva, B. Surer, S. Todo, S. Trebst, M. Troyer, M. L. Wall, P. Werner, and S. Wessel, “The ALPS project release 2.0: open source software for strongly correlated systems,” *Journal of Statistical Mechanics: Theory and Experiment* **2011**, P05001 (2011).
- [81] P. Jordan and E. Wigner, “Über das Paulische Äquivalenzverbot,” *Zeitschrift für Physik* **47**, 631–651 (1928).
- [82] Subir Sachdev, *Quantum Phase Transitions*, 2nd ed. (Cambridge University Press, Cambridge, United Kingdom, 2011).
- [83] J. Hubbard, “Electron correlations in narrow energy bands,” *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* **276**, 238–257 (1963).
- [84] Lei Wang, Philippe Corboz, and Matthias Troyer, “Fermionic quantum critical point of spinless fermions on a honeycomb lattice,” *New Journal of Physics* **16**, 103008 (2014).
- [85] Steven R. White and A. L. Chernyshev, “Neél order in square and triangular lattice heisenberg models,” *Phys. Rev. Lett.* **99**, 127004 (2007).
- [86] U. Schollwöck, “The density-matrix renormalization group,” *Rev. Mod. Phys.* **77**, 259–315 (2005).
- [87] L. Michel and I. P. McCulloch, “Schur Forms of Matrix Product Operators in the Infinite Limit,” *ArXiv e-prints* 1008.4667 (2010).
- [88] L. Mirsky, “A trace inequality of John von Neumann,” *Monatshefte für Mathematik* **79**, 303–306 (1975).
- [89] S. Geršgorin, “Über die Abgrenzung der Eigenwerte einer Matrix,” *Bulletin de l’Académie des Sciences de l’URSS. Classe des sciences mathématiques et naturelles*, 749–754 (1931).
- [90] Liqun Qi, “Some simple estimates for singular values of a matrix,” *Linear Algebra and its Applications* **56**, 105 – 119 (1984).

APPENDICES

Appendix A: Convenient features

OSMPS contains several features which exploit physical considerations to increase the power and reach of the algorithms or are not intrinsically based on physics but simplify the handling of simulations for the user. These convenient features are covered in this part of the appendix.

- **Symmetry conservation:** OSMPS supports an arbitrary number of $\mathcal{U}(1)$ symmetries. In order to employ symmetries the user has to provide the symmetry generator for each $\mathcal{U}(1)$ symmetry in a diagonal form. For the convergence study on the Bose-Hubbard model in Appendix B, the symmetry generator is the number operator n (which is diagonal). Therefore, simulations can be easily adapted to number conservation if possible. The advantage of the $\mathcal{U}(1)$ symmetries is the ensuing numerical speedup. It is comparable to the use of block-diagonal matrices versus the full matrix with the block diagonal structure. Decompositions are carried out on smaller subspaces. Taking advantage of the block diagonal

structure leads to a speedup of more than an order of magnitude for the example simulation treated in Table II.

Consider the one-dimensional Bose Hubbard model

$$H = -J \sum_{i=1}^{L-1} b_i b_{i+1}^\dagger + h.c. + \frac{1}{2} U \sum_{i=1}^L n_i (n_i - \mathbb{I}) - \mu \sum_{i=1}^L n_i, \quad (\text{A1})$$

where the chemical potential μ regulates the average number of particles $\bar{n} = \frac{1}{L} \sum_{i=1}^L \langle n_i \rangle$ in the non-number conserving case. We scan for unit filling with $\bar{n}(\mu = -0.22) \approx 1$ and use this chemical potential for the comparison together with $u = 1.0$ and $J = 0.5$. The compute times are determined from a *2x(Intel Xeon E5-2680 Dodeca-core) 24 Cores 2.50GHz* node. The Fortran library is compiled with *ifort* and optimization flag `O3` and using *mkl*. We see that the unit filling case can improve the simulation by an order of magnitude or more.

Settings	Time (s) without $\mathcal{U}(1)$	Time (s) with $\mathcal{U}(1)$
$\chi = 10, \varepsilon_V = 10^{-4}, \varepsilon_1 = 10^{-4}$	42	22
$\chi = 60, \varepsilon_V = 10^{-7}, \varepsilon_1 = 10^{-6}$	654	21
$\chi = 320, \varepsilon_V = 10^{-12}, \varepsilon_1 = 10^{-10}$	19838	150

Table II. Comparison of the CPU time for number conserving and non-number conserving MPS algorithm in the case of the Bose Hubbard model with local dimension $d = 6$ and $L = 32$ for different convergence parameters.

In addition, we study the Ising model defined in Eq. (15) where we can make use of the \mathbb{Z}_2 symmetry, which reduces the dimension of the Hilbert space by half. In the block diagonal structure we never have more than two blocks and we expect that using a symmetry-conserving MPS does not lead to the same speedup as in the Bose-Hubbard model. In Table III we find that for small bond dimension the overhead due to symmetry conservation is larger than the actual gain. This is in agreement with previous results in [12]. Starting with $\chi = 320$ in this table, the simulation with \mathbb{Z}_2 is faster than the one without symmetry. The simulations were run with the same setting as the Bose-Hubbard model simulations on a *2x(Intel Xeon E5-2680 Dodeca-core) 24 Cores 2.50GHz* node. The times include the ground state calculation, one local measurement, and one correlation measurement where the value of the external field $h = 0.98$ is close to the critical point.

Settings	Time (s) without \mathbb{Z}_2	Time (s) with \mathbb{Z}_2
$\chi = 10, \varepsilon_V = 10^{-4}, \varepsilon_1 = 10^{-4}$	0.15	0.79
$\chi = 60, \varepsilon_V = 10^{-7}, \varepsilon_1 = 10^{-6}$	0.57	1.94
$\chi = 320, \varepsilon_V = 10^{-12}, \varepsilon_1 = 10^{-10}$	6.31	5.81

Table III. Comparison of the CPU time for MPS ground states with and without conservation of \mathbb{Z}_2 symmetry in the quantum Ising model. The system size is $L = 128$, the interaction $J = 1$, the external field $h = 0.98$, and default convergence parameters are used if not specified in the table.

- **Support for fermionic systems:** Fermionic systems obey nonlocal anticommutation relations different from the local commutation relations of bosons. In order to represent fermionic operators in OSMPS, we use a Jordan-Wigner transformation [81, 82] which uses locally anticommuting operators together with strings of phase operators $(-1)^n$, with n the number operator. The routines within MPS, for example the measurement of correlation functions and construction of Hamiltonian terms, have a flag `Phase` for whether a string of phase operators is needed to enforce proper fermionic anticommutation relations.

Such fermions can be described via the Fermi-Hubbard model [83]. We consider in the following an example of spinless fermions using the Hamiltonian from [84] in a one-dimensional system

$$H_F = W \sum_{i=1}^{L-1} \left(n_i - \frac{1}{2} \right) \left(n_{i+1} - \frac{1}{2} \right) - J \sum_{i=1}^{L-1} \left(c_i^\dagger c_{i+1} + h.c. \right). \quad (\text{A2})$$

We briefly introduce the phase terms for fermionic systems and show that it affects the system. Instead of comparing a single correlation measurement or the single particle density matrix built from the correlations $\langle c_i^\dagger c_j \rangle$, we use the quantum depletion ξ from the eigenvalues Ξ_i of the single particle density matrix, yielding a single value. Furthermore, the quantum depletion ξ is constant for spinless fermions and errors can be detected more easily. If Ξ_i are the eigenvalues of the single particle density matrix, the depletion is defined as

$$\xi = 1 - \frac{\max_i \Xi_i}{\sum_i \Xi_i}. \quad (\text{A3})$$

Figure 12 shows multiple aspects of the model with a system size of $L = 65$ and approximately half-filling with 33 fermions in the system. The deviation from the average filling at each site

$$\delta = \frac{1}{65} \sum_{i=1}^{65} \left| n_i - \frac{33}{65} \right|, \quad (\text{A4})$$

and the bond entropy indicate the phase transition from a charge density wave to a superfluid phase dominated by the tunneling term in the Hamiltonian. Those values are plotted in Figure 12(a). In Figure 12(b) we show the quantum depletion in three different scenarios including calculating it correctly from the correlation with the phase terms, and incorrectly from the correlation without phase terms and the reduced density matrices. We emphasize that the reduced density matrices never contain phase terms and cannot be used to calculate correlation in fermionic systems which need a phase term.

- **MPI:** Message Passing Interface (MPI) is the standard for distributed memory parallel high performance computing. OSMPS supports the setup of data parallelism via MPI. Although data parallelism is the most basic implementation of a parallel algorithm, it represents the most efficient approach when iterating over a set of parameters. For a large fraction of problems we can assume that iterations over a set of parameters are necessary as in the case of phase diagrams, finite-size scalings, to analyze Kibble-Zurek scalings, or to scan over

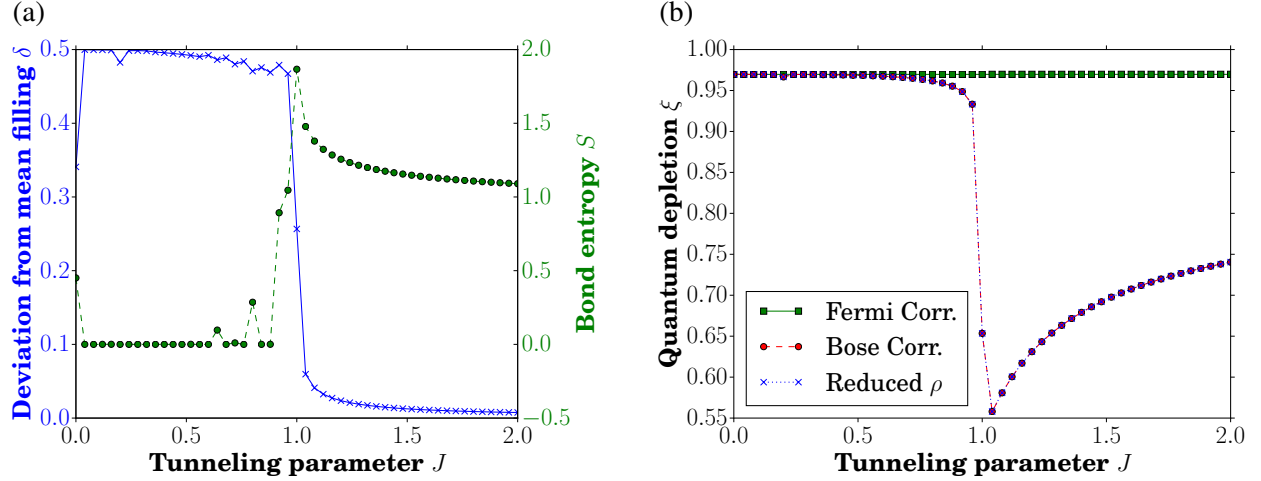


Figure 12. *Spinless Fermions at half-filling*. (a) The deviation from the mean-filling (blue x's) shows the two different phases. For small tunneling $J < 1$ there is an alternating order of empty-occupied sites dominating. In contrast, the tunneling term with $J > 1$ leads to a superfluid-like phase. The bond entropy shows the same phase transition with a peak around $J = 1.0$. We point out that the simulation for $J = 0$ fails because there are no fluctuations to feed the variational minimization in the ground state search. (b) The quantum depletion ξ is calculated from the correlation matrix $\langle c_i^\dagger c_j \rangle$. The result from the boson-type correlation without a phase term and the reduced density matrices are identical, corresponding to the lower curve. Both results do not match the correct curve when accounting for the phase terms of the Jordan-Wigner transformation.

a variety of initial conditions. We provide a Fortran implementation with one master and $(p - 1)$ workers where p is the total number of cores. Furthermore a Python implementation is available with p workers where one of them is distributing jobs.

As an example for the scaling of the MPI implementation we take a look at the Fortran implementation and the simulations necessary to generate the data for Fig. 2. We run the simulation on one or two nodes of the type *2x(Intel Xeon E5-2680 Dodeca-core) 24 Cores 2.50GHz* with 12, 24, 36, and 48 cores. The duration of the jobs can be found in Table IV. We recall that the set of simulations iterates over 30 data points for the system size L and 51 data points for the external field h , i.e., 1530 data points in total. In the case of 48 cores the average workload are approximately 32 data points. Since the master distributes the jobs to the workers one by one, each worker might handle less (more) jobs if their jobs take more (less) than the average time of one data point. In the setup of the simulation we avoid getting stuck in long simulations at the end having other cores running idle, because we address the large system sizes first. The longest single data point takes about 4.5 hours. This setup leads to a good scaling of the simulation time when increasing the number of cores. In fact, when increasing the number of cores from 12 by a factor of n , the speedup is greater than n . This is related to the fact that the master distributing the jobs has less weight for more cores. The parallel efficiency E_P is calculated with the cumulative CPU time of all simulations T_Σ and the actual run time of the job T_N on N cores. We assume that the cumulative CPU time corresponds to the time of the serial job:

$$E_P = \frac{T_\Sigma}{N \cdot T_N}. \quad (\text{A5})$$

The initial increase of E_P seen in Table IV is again explained with the master running almost idle.

Cores (Workers)	12 (11)	24 (23)	36 (35)	48 (47)	72 (71)	96 (95)
Time (hh:mm)	68:03	32:47	21:17	15:52	10:40	7:50
Efficiency E_P	91.7%	95.9%	97.3%	97.9%	97.1%	97.5%

Table IV. *MPI scaling for OSMPS*. As example for the MPI scaling we execute the simulations for Fig. 2 on different numbers of cores. In order to evaluate the scaling the number of workers excluding the master job distributing the jobs has to be considered.

- **Templates:** The OSMPS library supports calculations with real and complex numbers as well as standard MPS or symmetry conserving MPS algorithms inside the Fortran modules. The different data types lead to many redundant subroutines which we overcome by using templates. Subroutines are written for a generic type, e.g. `MPS_TYPE`. When generating the module the necessary types are plugged in, e.g. `MPS`, `MPS_C`, `qMPS`, and `qMPS_C`. The corresponding call to such a subroutine is covered under an interface (containing all the subroutines for the different types). These templates reduce errors copying from type to type and keep modules shorter.

Appendix B: Convergence studies

OSMPS can be tuned via multiple parameters modifying simulations with regards to the convergence. Therefore, we provide some guidelines for the convergence parameters along with examples. We divide this appendix in one part looking at the details of the finite size statics followed by additional studies of the time evolution methods.

1. Finite size variational algorithms

Due to the variational search we have several convergence parameters. We can divide them into three categories:

- *Lanczos*: Lanczos tolerance, maximal number of Lanczos iterations
- *Chi/bond dimension*: maximal bond dimension, local tolerance, variance tolerance
- *Iteration*: min/max num sweeps, max outer sweeps

For the convergence study presented in Table I we concentrate on the Lanczos tolerance ε_1 , the variance tolerance ε_V and the bond dimension χ . The maximal number of Lanczos iterations is set to a sufficiently high value to ensure convergence (500, default is 100). The number of inner sweeps is set to exactly 2, i.e. a minimum and maximum of 2, except for the spinless fermions with 4. We consider exactly one outer sweep. Since every additional outer sweep lowers the local tolerance $\varepsilon_{\text{local}}$ for the cut-off of singular values, we prevent different $\varepsilon_{\text{local}}$ depending on the number of outer sweeps carried out. The warmup phase to grow the system up to L sites has a warmup tolerance 100 times bigger than the variance tolerance and the warmup bond dimension

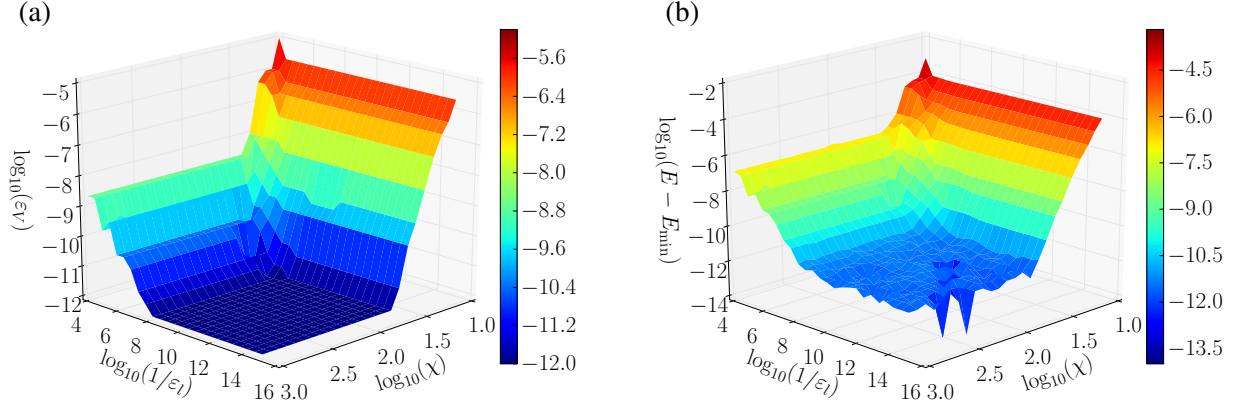


Figure 13. *Convergence study for the quantum Ising model.* (a) The boundary between converged and non-converged simulations as a function of the bond dimension and Lanczos tolerance. The surface shows the first converged simulations fulfilling the variance tolerance criteria. Smaller variance tolerances will lead to non-convergence for fixed parameters. With this plot we can estimate the maximal variance tolerance that can be achieved using a given bond dimension and Lanczos tolerance. (b) The energy difference as a function of bond dimension and Lanczos tolerance for the first converged simulation in regard to the variance tolerance (see (a)) reproduces the expectation that the energy difference decreases as the convergence criteria defined over the variance decreasing tolerance is met. Same labels apply to color bar and z -axis.

is half of the bond dimension. The local tolerance is then connected with the variance tolerance as specified in the default settings:

$$\epsilon_{\text{local}} = \frac{\epsilon_V}{4L}. \quad (\text{B1})$$

For the remaining three parameters ϵ_I , ϵ_V and χ the variance tolerance determines if a simulation is converged according to the criterion

$$\langle H^2 - \langle H \rangle^2 \rangle < \epsilon_V L. \quad (\text{B2})$$

The actual behavior may vary for different models. We study the convergence behavior for the Ising model, the Bose Hubbard model, and a spinless Fermi-Hubbard model. Future possibilities for similar studies include spinful Fermi-Hubbard or Bose-Hubbard models, XYZ models, quantum rotors, and disordered systems. Starting with the Ising model, we show in Fig. 13 (a) the boundary between converging and non-converging simulations over a grid of ϵ_I and χ . In Fig. 13 (b) the energy difference to the smallest value can be found. The point is close to the quantum critical point ($h = 0.98$ with $h_c = 1.0$) so the settings also serve as an upper bound for points further away from the critical point.

In the next example we turn to the long-range quantum Ising model with the Hamiltonian presented in Eq. (14). We evaluate the convergence behavior again close to the critical point. For a power-law decay with $\alpha = 3.0$ and a system size of $L = 128$ we have a critical field of $h_c \approx 1.35$ [47]. We leave the remaining parameter fixed in comparison to the nearest neighbor quantum Ising model. Figure 14 (a) shows again the boundary between converging and non-converging simulations based on the variance tolerance ϵ_V iterating over the bond dimension χ

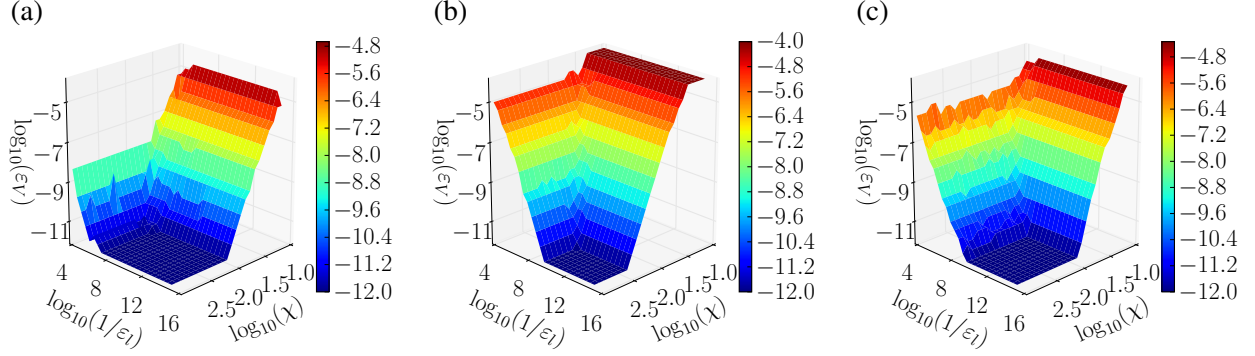


Figure 14. *Convergence study.* We study the variance tolerance achieved as a function of the bond dimension and the Lanczos tolerance for (a) the long-range Ising model, (b) the Bose-Hubbard model, and (c) a spinless Fermi-Hubbard model. (Same labels apply to color bar and z -axis.)

and the Lanczos tolerance ε_L . We see that we need a much higher bond dimension in comparison to the nearest-neighbor Ising model.

For the Bose Hubbard model introduced in Eq. (A1) the same type of plot is shown in Fig. 14(b). The algorithm includes number conservation at unit filling and the system size is $L = 32$. We choose the point with $U = 0.5$, $t = 0.5$ and $\mu = 0$ in the superfluid regime exhibiting long-range correlations. In contrast, the Mott insulator has less entanglement and therefore the superfluid parameters can serve as upper bound. In the Figs. 13 and 14 we observe that the Bose-Hubbard model needs, in comparison to the Ising model, stricter convergence parameters to arrive at an equal variance tolerance.

Finally, we consider the spinless fermions introduced in Eq. (A2) and present the result on the convergence in Fig. 14 (c). As with the Ising model, we choose a point close to the quantum critical point estimated in Fig. 12, that is a tunneling energy of $J = 1.04$. Furthermore, we take $L = 65$, and the system is filled with 33 fermions. In contrast to the other simulations we increase the minimum and maximum number of inner sweeps to 4. Comparing the Fermi-Hubbard model to the quantum Ising model since both have a local dimension of $d = 2$ and are nearest-neighbor, we see that the fermionic system needs a larger bond dimension χ in comparison.

2. Time evolution methods for finite size systems

In this part of the appendix we expand the convergence study of the time evolution methods with additional examples to demonstrate some key points. Figure 10 in Sec. V A shows aspects of a quench in the paramagnetic phase of the Ising model for $L = 10$. Due to the time-dependence of the Hamiltonian, we already make an error due to the evaluation of the Hamiltonian at discrete points in time. Now, we study the convergence of the different algorithms under the evolution of a time-independent Hamiltonian. Therefore, we use the same ground state of the Ising model at an external field $h = 5.0$ and evolve it under the Hamiltonian with an external field of $h = 4.5$, which corresponds to a sudden quench. The results are shown in Fig. 15.

We start the discussion with the maximal distance of the single site reduced density matrices.

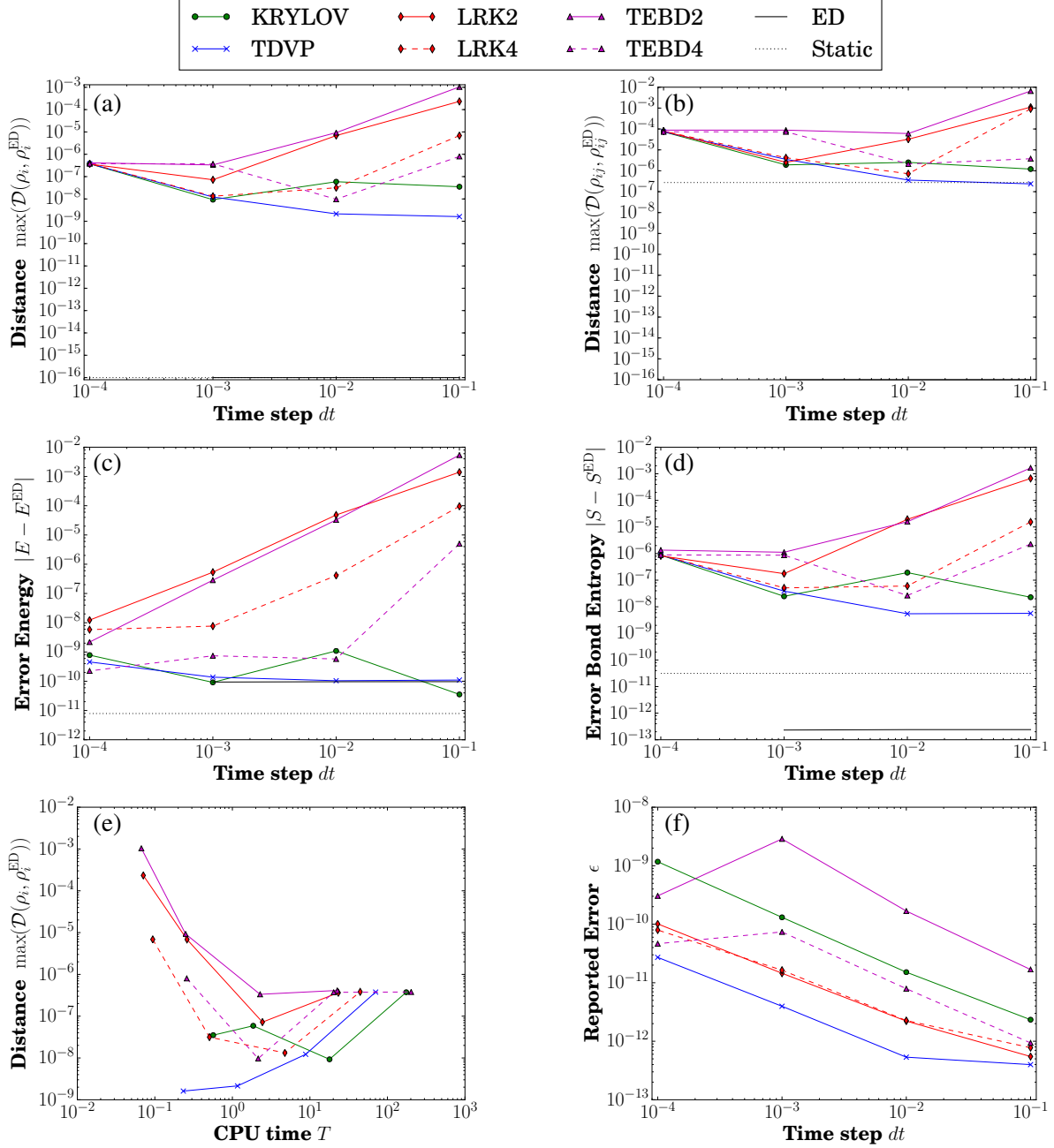


Figure 15. *Scaling of the error in time evolution methods* decreases as expected with the size of the time step for methods where the error depends on the time step. This is the example of a sudden quench of the Ising model in the paramagnetic phase evolving the ground state of an external field $h = 5.0$ at constant $h = 4.5$ for $L = 10$. Curves are a guide to the eye, points represent actual data.

For the Krylov and TDVP algorithms, the decrease of the time step dt does not improve the result, but seems to make it worse. Since there is no error in the method or from the time slicing of the Hamiltonian depending on dt , the number of applications is the critical variable to estimate the error. In contrast, TEBD and LRK have an error depending on dt in ϵ_{method} . Therefore the second order methods are worse than the fourth order and smaller time steps improve the result if it did

not yet reach the lower bound. We recall that the lower bound is considered to be error between the ground state results of MPS and exact diagonalization. The two-site reduced density matrices, the energy, and the bond entropy support this trend. Further the errors in energy give an indication to judge on the rate of convergence for LRK and TEBD. If we consider the data points for $dt = 0.1$ and $dt = 0.01$, the slope for the TEBD4 curve is bigger than the one for TEBD2 indicating a better rate of convergence. Then TEBD4 reaches the lower bound induced by the error of the initial state or is at the level of the error of the TDVP and Krylov method. The LRK methods have the same rate of convergence following this argumentation. The look at the CPU times yields then a counterintuitive result. The most precise simulation with TDVP has the biggest time step and one of the shortest run times. The reported error from OSMPS follows the arguments from the time-dependent Hamiltonian. Without any truncation besides the local tolerance, more time steps add up to more error contributions.

Appendix C: Scaling of computational resources

We consider the scaling of computational resources, especially computation time and memory, as a function of common parameters of simulations. These include the system size L and the maximal bond dimension χ . While the default parallelization is data parallelism using the MPI interface to OSMPS with a straightforward scaling explained in Appendix A, we consider as well the openMP (Open Multi Processing) algorithms used by the underlying libraries, namely LAPACK and BLAS.

First, we consider the scaling with the maximal bond dimension χ . We consider in this scenario the quantum Ising model for a system size of $L = 1063$, which corresponds to one data point from Fig. 2. The value of the external field is the critical value for the thermodynamic limit, i.e., $h = 1.0$. The simulations were run on a *2x(Intel Xeon E5-2680 Dodeca-core) 24 Cores 2.50GHz* node and the corresponding data is shown in Fig. 16. In order to have a better understanding of the data we plot in each case the bond dimension. For the memory in Fig. 16(a) we see that the file size saturates once the bond dimension saturates. The file contains the complete information about the state in binary format and gives a good estimate of the memory needs for each simulation. For linear algebra operations within LAPACK, the additional memory allocated as workspace is on the order of the matrix size. The size of the matrices handled is bounded by the local dimension and the maximal bond dimension leading to a maximum size of $d_{\chi_{\max}} \times d_{\chi_{\max}}$. Figure 16(b) shows the CPU time for each simulation. It naturally saturates with the bond dimension. Before the saturation point it grows linearly with the bond dimension actually used.

For the scaling of the resources with the system size L we consider the set of simulations generating Fig. 2 and pick the external field $h = 1.0$. The data is for *Penguin Relion 2x(Intel X5675) 12 cores 3.06GHz* nodes. As previously, we show the bond dimension utilized in addition to the file size or computation time. If the bond dimension saturates starting at $L > 500$, the file size grows linearly with the system size L as shown in Fig. 16(c). In contrast, in the growth for $L < 500$ the file size increases faster than linear. In addition to the growing system size, the fact that a larger system can have more entanglement leads to the increase in file size. Figure 16(d) shows the CPU times for the equivalent setup with a similar result as for the file size. The scaling is linear as soon as the bond dimension actually used saturates. Before the growth appears nonlinear with a jump.

We examine the use of openMP as the last part of the analysis of resources. OpenMP allows for parallel computing with shared memory on one compute node. In contrast, the tasks of MPI jobs never have access to the same memory and have to send any data. In our type of implementation

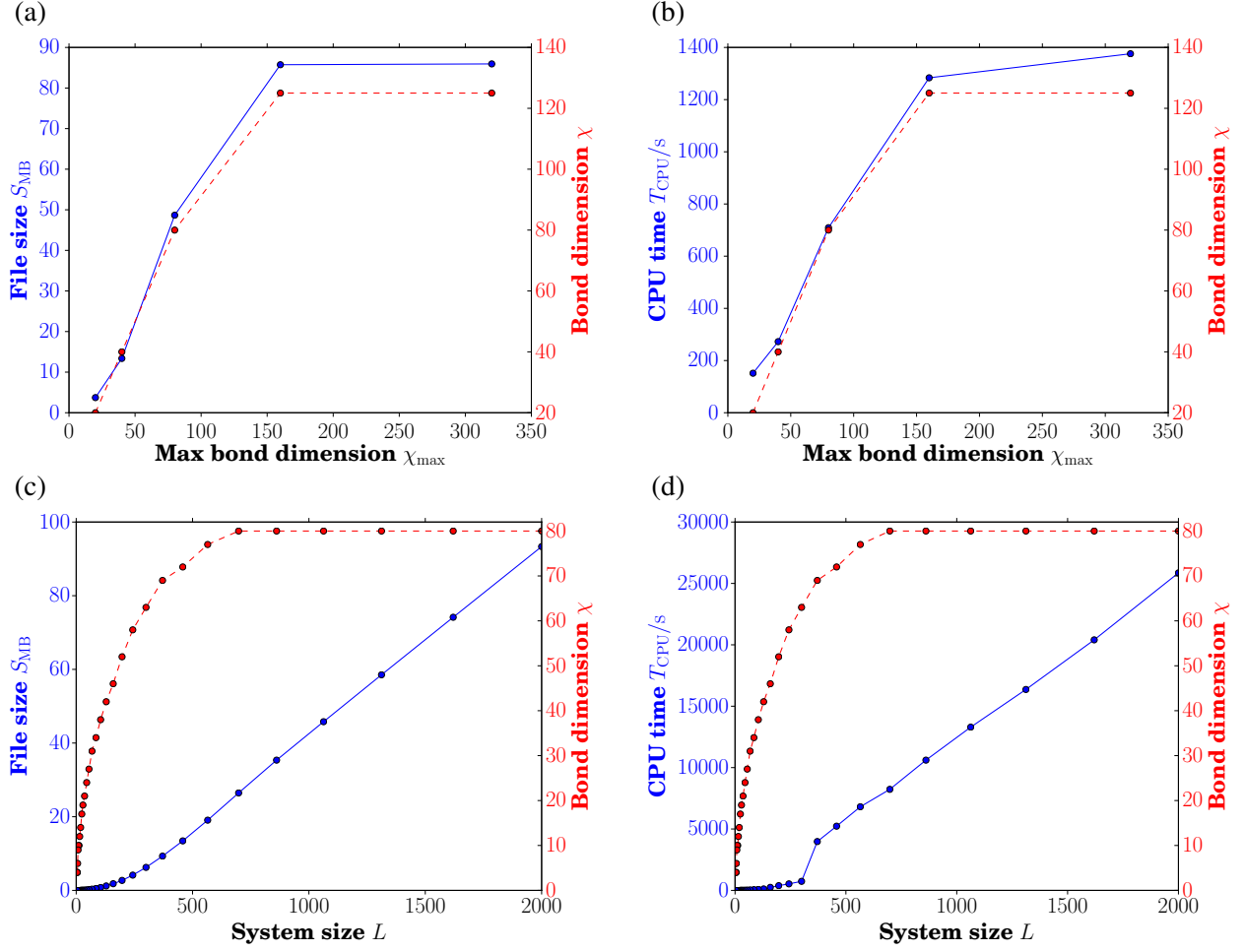


Figure 16. *Scaling of computational resources.* We consider the Ising model and vary the bond dimension and the system size. (a) The file size of the ground state indicates the memory resources necessary for the simulation. The parameters are $L = 1063$ and $h = 1.0$ for different maximal bond dimensions χ_{max} . (b) The scaling of the CPU time as a function of χ_{max} . Parameters equal to (a). (c) The file size of the ground state for $\chi_{\text{max}} = 80$ at $h = 1.0$ for a set of different system sizes from the data of Fig. 2. (d) The scaling of the CPU time for different system sizes. Parameters equal to (c).

data is sent between the master node and the workers. On current clusters this allows one to use parallelization with up to 24 cores. We study the efficiency of openMP and what speedup can be gained. OSMPS does not have implementations for openMP itself, but the LAPACK and BLAS or respectively mkl libraries can support openMP on the level of the linear algebra operations within OSMPS. We find that openMP, implemented in this form, does not increase the speed of the simulation, as described in Table V. The simulation time shown is the duration of the job and not the CPU time on a $2 \times (\text{Intel Xeon E5-2680 Dodeca-core})$ 24 Cores 2.50GHz node.

Cores	1	2	4	8	16	24
$T(\chi_{\max} = 20)/s$	150	154	154	155	154	168
$T(\chi_{\max} = 40)/s$	270	273	271	274	275	302
$T(\chi_{\max} = 80)/s$	699	780	789	734	749	739
$T(\chi_{\max} = 160)/s$	1293	1273	1232	1188	1161	1396
$T(\chi_{\max} = 3200)/s$	1368	1372	1259	1241	1237	1278

Table V. *OpenMP scaling for OSMPS*. As an example for the scaling we simulate the ground state of the Ising model for $L = 1063$ at an external field of $h = 1.0$ for different number of threads and different bond dimensions. This case corresponds to a data point of Fig 2. We see that openMP does not come with any real speedup. The maximal bond dimension used is 124, which affects the last two rows.

Appendix D: Building and installing the open source Matrix Product States library

In order to install OSMPS, one needs to provide at a minimum the following packages covering the Fortran and Python installation and linear algebra tools:

- Python (at least 2.6 or 2.7 recommended, 3.x tested successfully)
- numpy and scipy
- Fortran2003 compiler, e.g. *gfortran* for Linux.
- BLAS and LAPACK
- ARPACK

More packages might be convenient e.g. for plotting with Python, but are not mandatory. The remaining part of the installation is covered by Python. We distinguish between a global and a local installation where the local installation can only be accessed in the same directory or through providing the path. For global installation only one must execute on the command line

```
sudo python setup.py install
```

The Fortran library is compiled then with the command

```
python BuildOSMPS.py -os=unixmpi
```

with prefix `sudo` for the global installation in order to provide the admin privileges. Some default settings are available via command line options as shown for compiling with unix and MPI. Similarly, a local installation can be specified with the option `--local='./'`. More specific settings can be made inside the Python script, which builds the makefile for the Fortran libraries.

Appendix E: User support and contributing to the code

Being an open source project we welcome anyone willing to help improve OSMPS or share her knowledge with the library. On the project website <http://sourceforge.net/projects/>

[openmps/](#) on SourceForge we maintain a forum for discussions about OSMPS. These discussions include questions about the installation and use of the algorithms for new users, suggestions for future implementations, and help requests for implementing new features on your own.

The current version of OSMPS is organized via the *svn* version control system via SourceForge. It is possible without a user account to fetch the newest version of the library via *svn*. In order to contribute to OSMPS, a user account on *SourceForge* is necessary so that we can add you to the developers team.

Appendix F: Error bounds for static simulations

Ideally numerical simulations yield a corresponding bound for the error of their results. For DMRG there are calculations available discussing the behavior of the error. Local observables in a two-dimensional Heisenberg models are discussed in terms of the truncation error in [8, 85, 86]. The error in variational MPS methods is mentioned in [64, 87] and relates the variance to the squared norm of the difference between the exact and the approximate quantum state. We present an alternate approach using the variance as well to derive error expressions for multiple observables.

In static MPS simulations the variance of the Hamiltonian, which bounds the error of the energy, is returned as an error estimate for the result. Therefore, it remains for us to show how other observables or measures are bounded by the variance of the state, where we take the ground state as an example. The basic idea is to assume we have a final state $|\psi\rangle$ as an MPS

$$|\psi\rangle = f|\psi_0\rangle + \epsilon|\psi_\perp\rangle, \quad (\text{F1})$$

with $f = \langle\psi|\psi_0\rangle$, $|\epsilon|^2 = 1 - |f|^2$, and $|\psi_0\rangle$ the true ground state, while $|\psi_\perp\rangle$ is orthogonal to the ground state and contains all errors. In this appendix we keep the notation that $|\psi\rangle$ is the state from the OSMPS simulation with a variance V_ψ , $|\psi_0\rangle$ is the true ground state and $|\psi_\perp\rangle$ contains all contributions orthogonal to the true ground state.

1. Bounding ϵ with the variance delivered by open source Matrix Product States

The first step is to bound ϵ from the information gathered in OSMPS, that is the variance of H . Furthermore, in addition to the above orthogonality relation $\langle\psi_\perp|\psi_0\rangle = 0$, any power H^n is subject to the relation $\langle\psi_\perp|H^n|\psi_0\rangle = 0$, since $|\psi_0\rangle$ is an eigenstate of H . We recall that the Hamiltonian is represented without errors except if an `InfiniteFunction` rule is fitted through a series of `Exponential` rules. The error due such a fitting procedure is not covered in this Appendix. Writing the variance in terms of this decomposition we obtain

$$\begin{aligned} V_\psi &= \langle\psi|H^2|\psi\rangle - (\langle\psi|H|\psi\rangle)^2 \\ &= |f|^2 \langle\psi_0|H^2|\psi_0\rangle + f^*\epsilon \langle\psi_0|H^2|\psi_\perp\rangle + f\epsilon^* \langle\psi_\perp|H^2|\psi_0\rangle + |\epsilon|^2 \langle\psi_\perp|H^2|\psi_\perp\rangle \\ &\quad - (|f|^2 \langle\psi_0|H|\psi_0\rangle + f^*\epsilon \langle\psi_0|H|\psi_\perp\rangle + f\epsilon^* \langle\psi_\perp|H|\psi_0\rangle + |\epsilon|^2 \langle\psi_\perp|H|\psi_\perp\rangle)^2 \\ &= |f|^2 \langle\psi_0|H^2|\psi_0\rangle + |\epsilon|^2 \langle\psi_\perp|H^2|\psi_\perp\rangle \\ &\quad - (|f|^4 \langle\psi_0|H|\psi_0\rangle^2 + 2|f|^2|\epsilon|^2 \langle\psi_0|H|\psi_0\rangle \langle\psi_\perp|H|\psi_\perp\rangle + |\epsilon|^4 \langle\psi_\perp|H|\psi_\perp\rangle^2). \quad (\text{F2}) \end{aligned}$$

We introduce the eigenenergy E_0 of our true ground state and the energy E_\perp , which is not an eigenenergy of the system because $|\psi_\perp\rangle$ can be a linear combination of eigenstates. Next, we

use the relation $|f|^2 = 1 - |\epsilon|^2$ and we add a zero in terms of $\pm(|\epsilon|^2 - |\epsilon|^4)E_\perp^2$ to simplify the expression:

$$\begin{aligned} V_\psi &= (1 - |\epsilon|^2) \langle \psi_0 | H^2 | \psi_0 \rangle + |\epsilon|^2 \langle \psi_\perp | H^2 | \psi_\perp \rangle \\ &\quad - (1 - 2|\epsilon|^2 + |\epsilon|^4) E_0^2 - 2(|\epsilon|^2 - |\epsilon|^4) E_0 E_\perp - |\epsilon|^4 E_\perp^2 \\ &\quad + (|\epsilon|^2 - |\epsilon|^4) E_\perp^2 - (|\epsilon|^2 - |\epsilon|^4) E_\perp^2 \\ &= (1 - |\epsilon|^2) V_0 + |\epsilon|^2 V_\perp + (|\epsilon|^2 - |\epsilon|^4) (E_\perp - E_0)^2. \end{aligned} \quad (\text{F3})$$

We abbreviate the energy difference as $\Delta = E_\perp - E_0$, define the variance of $|\psi_\perp\rangle$ as V_\perp , and the variance of $|\psi_0\rangle$ as $V_0 = 0$. We introduce $\tilde{\epsilon} = |\epsilon|^2$ leading to the following quadratic equation:

$$\Delta^2 \tilde{\epsilon}^2 - (V_\perp + \Delta^2) \tilde{\epsilon} + V_\psi = 0. \quad (\text{F4})$$

We remark that the variance of an eigenstate of H is zero, applied to $V_0 = 0$. The equation has two solutions returning the values for $|\epsilon|^2$:

$$|\epsilon_1|^2 = \frac{1}{2} \left(1 + \frac{V_\perp}{\Delta^2} - \sqrt{\left(1 + \frac{V_\perp}{\Delta^2} \right)^2 - 4 \frac{V_\psi}{\Delta^2}} \right) \quad (\text{F5})$$

$$|\epsilon_2|^2 = \frac{1}{2} \left(1 + \frac{V_\perp}{\Delta^2} + \sqrt{\left(1 + \frac{V_\perp}{\Delta^2} \right)^2 - 4 \frac{V_\psi}{\Delta^2}} \right). \quad (\text{F6})$$

Plugging $|\epsilon|^2 = 1 - |f|^2$ into Eq. (F4), we get another quadratic equation and two solutions for $|f|^2$:

$$\Delta^2 |f|^4 - (\Delta^2 - V_\perp) |f|^2 + (V_\psi - V_\perp) = 0, \quad (\text{F7})$$

$$|f_1|^2 = \frac{1}{2} \left(1 - \frac{V_\perp}{\Delta^2} + \sqrt{\left(1 + \frac{V_\perp}{\Delta^2} \right)^2 - \frac{4V_\psi}{\Delta^2}} \right), \quad (\text{F8})$$

$$|f_2|^2 = \frac{1}{2} \left(1 - \frac{V_\perp}{\Delta^2} - \sqrt{\left(1 + \frac{V_\perp}{\Delta^2} \right)^2 - \frac{4V_\psi}{\Delta^2}} \right). \quad (\text{F9})$$

According to the normalization condition we recognize that f_1 and ϵ_1 build one solution, as well as f_2 and ϵ_2 :

$$|f_i|^2 + |\epsilon_i|^2 = 1, \quad i \in \{1, 2\}. \quad (\text{F10})$$

Under the assumption that the major part of our state is in the ground state, we choose the smaller ϵ_1 . This is the first assumption in the calculation and we proceed to bound ϵ^2 using the fact that $0 \leq V_\perp \leq V_\psi$ and then implementing a Taylor expansion in V_ψ/Δ , which is small if the variance of the ground state has sufficiently converged targeting in the default setup a value of $L \times 10^{-10}$ and we have a energy gap in the system. We point out the role of the gap Δ in the following steps in detail:

$$\begin{aligned} |\epsilon|^2 &\leq \frac{1}{2} \left(1 + \frac{V_\psi}{\Delta^2} - \sqrt{1 - 4 \frac{V_\psi}{\Delta^2}} \right) \\ &\approx \frac{1}{2} \left(1 + \frac{V_\psi}{\Delta^2} - \left(1 - 2 \frac{V_\psi}{\Delta^2} - \mathcal{O} \left(\frac{V_\psi^2}{\Delta^4} \right) \right) \right) = \frac{1}{2} \left(\frac{3V_\psi}{2\Delta^2} + \mathcal{O} \left(\frac{V_\psi^2}{\Delta^4} \right) \right). \end{aligned} \quad (\text{F11})$$

Finally, we use the minimal gap between the ground state and the first excited state $\Delta_{0,1} = E_1 - E_0$ to approximate Δ . The inverse energy difference between the ground state and a superposition of all excited states can be bound with the smallest gap:

$$|\epsilon|^2 \leq \frac{3V_\psi}{4\Delta_{0,1}^2} + \mathcal{O}\left(\frac{V_\psi^2}{\Delta_{0,1}^4}\right) \implies |\epsilon| \leq \frac{\sqrt{V_\psi}}{\Delta_{0,1}} + \mathcal{O}\left(\frac{V_\psi}{\Delta_{0,1}^2}\right). \quad (\text{F12})$$

This bound shows that the variance can be a good approximation for the error as long as the gap to the next eigenstate is finite. This gap refers to the next accessible eigenstate in case symmetries are used. For simulations around the critical point with a closing gap the bound becomes less precise due to the closing gap. But for finite systems considered in this calculation, the system remains with a gap.

In addition we list the implicit and explicit assumptions during the derivation of the bound

- We converged mainly to the ground state. The solutions in Eqs. (F5) and (F7) and in the Eqs. (F6) and (F9) are in general true for any eigenstate of H .
- By taking the solution represented by the pair of Eqs. (F5) and (F7) we assume that the main part of the solution is in the eigenstate.
- The energy gap to the first state above the ground state used in the Taylor expansion is assumed not to be small. This approximation may fail around quantum critical points with a closing energy gap.
- $V_\perp \leq V_\psi$: the variance of the subset of states is smaller if the minimal energy is canceled from the set of states. This is true since one end of the distribution is cut.

2. Bounding observables

The ϵ derived above is only useful if bounds for other measures can be obtained. For a local Hermitian observable O with a maximal absolute value M defined as

$$\mathcal{M} = \max_{|\phi\rangle, |\phi'\rangle} |\langle \phi | O | \phi' \rangle| \quad (\text{F13})$$

we obtain the following bound between the measurement on the true ground state $|\psi_0\rangle$ and the state $|\psi\rangle$ resulting from the OSMPS simulation:

$$\begin{aligned} |\langle \psi_0 | O | \psi_0 \rangle - \langle \psi | O | \psi \rangle| &= |(1 - |f|^2) \langle \psi_0 | O | \psi_0 \rangle - \epsilon f^* \langle \psi_0 | O | \psi_\perp \rangle - \epsilon^* f \langle \psi_\perp | O | \psi_0 \rangle \\ &\quad - |\epsilon|^2 \langle \psi_\perp | O | \psi_\perp \rangle| \\ &\leq (1 - |f|^2) |\langle \psi_0 | O | \psi_0 \rangle| + 2|\epsilon f| |\langle \psi_0 | O | \psi_\perp \rangle| + |\epsilon|^2 |\langle \psi_\perp | O | \psi_\perp \rangle| \\ &\leq 2|\epsilon|(|f| + |\epsilon|)\mathcal{M} \leq 2\sqrt{2}|\epsilon|\mathcal{M} \leq 3|\epsilon|\mathcal{M}. \end{aligned} \quad (\text{F14})$$

Here in, $|f| + |\epsilon|$ is always smaller than $\sqrt{2}$, which originates in the normalization of the state and can be derived from the maximizing the constraint problem $\mathcal{L}(f, \epsilon, \gamma) = |f| + |\epsilon| + \gamma(|f|^2 + |\epsilon|^2 - 1)$, where γ is a Lagrange multiplier for the normalization constraint. So in general it is possible to bound values of an observable.

3. Density matrices and their bounds

In the following, we derive an expression for the complete density matrix, any reduced density matrix of the system, and a bound for the trace distance between the MPS result and the (reduced) density matrix of the ground state. We start by expressing the mixed contribution of the form $|\psi_0\rangle\langle\psi_\perp|$ in a more convenient way.

Lemma 1 (Mixed contributions $|\psi_0\rangle\langle\psi_\perp|$). *A mixed contribution of the form $e^{-i\phi}|\psi_0\rangle\langle\psi_\perp| + e^{i\phi}|\psi_\perp\rangle\langle\psi_0|$ with an arbitrary phase ϕ can be decomposed into positive matrices σ_\pm with one non-zero eigenvalue of value 1 as*

$$|\psi_0\rangle\langle\psi_\perp| + |\psi_\perp\rangle\langle\psi_0| = \sigma_+ - \sigma_-, \quad (\text{F15})$$

where σ_\pm fulfill all characteristics of a density matrix (and are not the spin lowering/raising operators).

Proof. We take the pure states $|\psi_\pm\rangle = (|\psi_0\rangle \pm e^{i\phi}|\psi_\perp\rangle)/\sqrt{2}$, where ϕ is an arbitrary phase. These pure states define the density matrices σ_\pm :

$$\sigma_\pm = |\psi_\pm\rangle\langle\psi_\pm| = \frac{1}{2}|\psi_0\rangle\langle\psi_0| + \frac{1}{2}|\psi_\perp\rangle\langle\psi_\perp| \pm \frac{1}{2}(e^{-i\phi}|\psi_0\rangle\langle\psi_\perp| + e^{i\phi}|\psi_\perp\rangle\langle\psi_0|). \quad (\text{F16})$$

Therefore, the difference $\sigma_+ - \sigma_-$ leads to the term we need while canceling out the contributions from $|\psi_\perp\rangle\langle\psi_\perp|$ and $|\psi_0\rangle\langle\psi_0|$:

$$\sigma_+ - \sigma_- = e^{-i\phi}|\psi_0\rangle\langle\psi_\perp| + e^{i\phi}|\psi_\perp\rangle\langle\psi_0|. \quad (\text{F17})$$

Lemma 2 (Error bound on density matrix). *Knowing ϵ in $|\psi\rangle = f|\psi_0\rangle + \epsilon|\psi_\perp\rangle$, we can express the (reduced) density matrix ρ representing the OSMPS result as*

$$\rho = |f|^2\rho_0 + \tilde{\epsilon}_+\rho_+ - \tilde{\epsilon}_-\rho_-, \quad \tilde{\epsilon}_\pm < 2|\epsilon|, \quad (\text{F18})$$

where ρ_0 is the density matrix of the exact ground state and ρ_\pm are density matrices containing the error.

Proof. We build the density matrix ρ on the complete Hilbert space as

$$\begin{aligned} \rho &= |\psi\rangle\langle\psi| = |f|^2|\psi_0\rangle\langle\psi_0| + f\epsilon^*|\psi_0\rangle\langle\psi_\perp| + f^*\epsilon|\psi_\perp\rangle\langle\psi_0| + |\epsilon|^2|\psi_\perp\rangle\langle\psi_\perp| \\ &= |f|^2|\psi_0\rangle\langle\psi_0| + |f\epsilon|(e^{-i\phi}|\psi_0\rangle\langle\psi_\perp| + e^{i\phi}|\psi_\perp\rangle\langle\psi_0|) + |\epsilon|^2|\psi_\perp\rangle\langle\psi_\perp|, \end{aligned} \quad (\text{F19})$$

where we have chosen the phase ϕ such that $f\epsilon^* = |f\epsilon|e^{-i\phi}$. From Lemma 1 we obtain

$$\begin{aligned} \rho &= |f|^2\rho_0 + |\epsilon f|(\sigma_+ - \sigma_-) + |\epsilon|^2\rho_\perp = |f|^2\rho_0 + |\epsilon|(|f|\sigma_+ + |\epsilon|\rho_\perp) - |f|\sigma_- \\ &= |f|^2\rho_0 + |\epsilon|\left((|f| + |\epsilon|)\frac{|f|\sigma_+ + |\epsilon|\rho_\perp}{|f| + |\epsilon|} - |f|\sigma_-\right). \end{aligned} \quad (\text{F20})$$

Defining the density matrices with positive and negative sign, we obtain

$$\rho = |f|^2\rho_0 + \tilde{\epsilon}_+\rho_+ - \tilde{\epsilon}_-\rho_-, \quad \tilde{\epsilon}_+ = |\epsilon|(|f| + |\epsilon|) \leq 2|\epsilon|, \quad \tilde{\epsilon}_- = |\epsilon f| \leq |\epsilon|, \quad (\text{F21})$$

$$\rho_+ = \frac{|f|\sigma_+ + |\epsilon|\rho_\perp}{|f| + |\epsilon|}, \quad \rho_- = \sigma_-, \quad (\text{F22})$$

where all matrices $\rho_{0,\pm}$ are density matrices with trace 1 and fulfilling positivity.

Lemma 3 (Bound on reduced density matrices). *The bound on the reduced density matrices ρ_A of the OSMPS result is equal to the bound on the complete density matrix, which is*

$$\rho_A = \text{Tr}_B \rho = |f|^2 \rho_{0,A} + \tilde{\epsilon}_+ \rho_{+,A} - \tilde{\epsilon}_- \rho_{-,A}, \quad (\text{F23})$$

where we consider an arbitrary bipartition of our system in parts A and B , tracing out over the bipartition B .

Proof. In order to obtain a reduced density matrix, we define subsystem A and B and take the partial trace over subsystem B :

$$\rho_A = \text{Tr}_B \rho = \text{Tr}_B (|f|^2 \rho_0 + \tilde{\epsilon}_+ \rho_+ - \tilde{\epsilon}_- \rho_-). \quad (\text{F24})$$

As a linear operation the previous expression can be rewritten as

$$\rho_A = |f|^2 \text{Tr}_B \rho_0 + \tilde{\epsilon}_+ \text{Tr}_B \rho_+ - \tilde{\epsilon}_- \text{Tr}_B \rho_- = |f|^2 \rho_{0,A} + \tilde{\epsilon}_+ \rho_{+,A} - \tilde{\epsilon}_- \rho_{-,A}. \quad (\text{F25})$$

4. Bound for the trace distance

Now that we are able to bound the density matrices with regards to the density matrix of the exact ground state, we can continue to prove a bound expressed in the trace distance \mathcal{D} .

Lemma 4 (Bound on the trace distance). *The trace distance between the density matrix ρ from OSMPS and the density matrix of the true ground state ρ_0 can be bounded with*

$$\mathcal{D}(\rho, \rho_0) \leq \frac{\sqrt{2V_\psi}}{\Delta_{0,1}}. \quad (\text{F26})$$

Proof. We continue with a bound on the trace distance defined as

$$\mathcal{D}(\rho, \rho_0) = \frac{1}{2} \text{Tr} \sqrt{(\rho - \rho_0)^\dagger (\rho - \rho_0)} = \frac{1}{2} |\rho - \rho_0|, \quad (\text{F27})$$

where the simplification can be made due to the Hermitian property of the density matrices. We first concentrate on expressing the difference between the density matrices in a convenient way using the expressions for $\tilde{\epsilon}_\pm$ in Eq. (F21):

$$\begin{aligned} \rho - \rho_0 &= |f|^2 \rho_0 + \tilde{\epsilon}_+ \rho_+ - \tilde{\epsilon}_- \rho_- - \rho_0 = \tilde{\epsilon}_+ \rho_+ - \tilde{\epsilon}_- \rho_- - |\epsilon|^2 \rho_0 = \tilde{\epsilon}_+ \rho_+ - |\epsilon| (|f| \rho_- + |\epsilon| \rho_0) \\ &= \tilde{\epsilon}_+ (P - M), \quad P \equiv \rho_+, \quad M \equiv \frac{|f| \rho_- + |\epsilon| \rho_0}{|f| + |\epsilon|}. \end{aligned} \quad (\text{F28})$$

The new matrices P and M are again defined so that they fulfill the requirements for a density matrix (Hermitian, positive, trace equal to 1). In the next step we make use of the triangular inequality of the trace norm:

$$\begin{aligned} \mathcal{D}(\rho, \rho_0) &= \frac{1}{2} \text{Tr} |\rho - \rho_0| = \frac{|\epsilon|(|f| + |\epsilon|)}{2} \text{Tr} |P - M| \leq \frac{|\epsilon|(|f| + |\epsilon|)}{2} (\text{Tr} |P| + \text{Tr} |M|) \\ &= |\epsilon|(|f| + |\epsilon|) \leq \sqrt{2} |\epsilon| \leq \frac{\sqrt{2V_\psi}}{\Delta_{0,1}}. \end{aligned} \quad (\text{F29})$$

5. Bound on the bond entropy

In order to get the bound on the bond entropy, we use Fannes' inequality [57] stating that the difference of the entropy S of two density matrices of dimension $D \times D$ is bounded by the trace distance \mathcal{D} between those two density matrices:

$$|S(\rho) - S(\sigma)| \leq \mathcal{D} \log_a(D) - \mathcal{D} \log_a(\mathcal{D}), \quad \forall \mathcal{D} \leq \frac{1}{e} \approx 0.36, \quad (\text{F30})$$

with the logarithm for the von Neumann entropy base a and D the dimension of the Hilbert space for the density matrices ρ and σ . The inequality was derived for logarithm base two, $a = 2$, but holds for any basis. Therefore, the bound for the bond entropy is

$$\epsilon_S \equiv |S(\rho_0) - S(\rho)| \leq \frac{\sqrt{2V_\psi}}{\Delta_{0,1}} \log(D) - \frac{\sqrt{2V_\psi}}{\Delta_{0,1}} \log\left(\frac{\sqrt{2V_\psi}}{\Delta_{0,1}}\right). \quad (\text{F31})$$

Appendix G: Bounding measurement with the trace distance

We have used the trace distance of reduced density matrices to judge the convergence of our results. We now show that the trace distance bounds from above any observable defined on the reduced density matrix. As a preliminary result we need that the absolute value of the eigenvalues of a Hermitian matrix correspond to the singular values of the matrix. Therefore, we use the fact that the matrix can be decomposed via an eigenvalue decomposition $A = U\Lambda U^\dagger$ and a singular value decomposition $A = \tilde{U}S\tilde{V}^\dagger$. Calculating AA^\dagger with both expressions leads to

$$AA^\dagger = U\Lambda^2U^\dagger = \tilde{U}S^2\tilde{U}^\dagger, \quad (\text{G1})$$

which shows that the absolute values of the eigenvalues Λ are equal to the singular values S . We use the fact that the absolute values of the eigenvalues of a Hermitian matrix are the singular values in the following when estimating the difference between the measurement of an observable Q for two different density matrices ρ and σ defined as

$$\epsilon(\rho, \sigma, Q) = |\text{Tr}(Q\rho) - \text{Tr}(Q\sigma)| = |\text{Tr}(Q(\rho - \sigma))|. \quad (\text{G2})$$

In order to bound this with the trace distance of ρ and σ we use the von Neumann trace inequality [88] on the matrices Q and $(\rho - \sigma)$ leading to

$$\epsilon(\rho, \sigma, Q) \leq \sum_i^D \kappa_i |\Lambda_i|, \quad (\text{G3})$$

where κ_i are the singular values of Q sorted in descending order with $\kappa_i > \kappa_{i+1}$. The Λ_i are the eigenvalues of $(\rho - \sigma)$ sorted in descending order with $|\Lambda_i| > |\Lambda_{i+1}|$. D is the dimension of the density matrix and the corresponding Hilbert space. We derived the relation Eq. (G1) for this purpose. We approximate Eq. (G3) by choosing the maximal singular values κ_{\max} from all κ_i :

$$\epsilon(\rho, \sigma, Q) \leq \kappa_{\max} \sum_i |\Lambda_i| = \kappa_{\max} \text{Tr} |\Lambda_i| = \kappa_{\max} 2\mathcal{D}(\rho, \sigma). \quad (\text{G4})$$

For further convenience we estimate κ_{\max} as a function of the observable Q . In the case of a Hermitian observable $Q = Q^\dagger$ we use again the connection between eigenvalues and singular

values from Eq. (G1). The eigenvalues of a square matrix can be estimated with Geršgorin circles [63, 89, 90]. Each Geršgorin circle has its origin in the diagonal element of the matrix; the radius is the sum of the absolute values of the non-diagonal of the corresponding row (or column). In this case we are only interested in the absolute values and the expression for κ_{\max} in terms of the Geršgorin circles simplifies to

$$\kappa_{\max} \leq \max_i \left(\sum_j |Q_{i,j}| \right). \quad (\text{G5})$$

In the case of a non-Hermitian Q , i.e., the correlation measurement $\langle b_i b_j^\dagger \rangle$, we can obtain an estimate using the fact that the singular values are connected to the eigenvalues of QQ^\dagger . If Q has the singular value decomposition $Q = USV^\dagger$, then QQ^\dagger has the eigenvalue decomposition $QQ^\dagger = US^2U^\dagger$. Therefore, the square root of the positive eigenvalues of QQ^\dagger are the singular values of Q . For the non-Hermitian Q we can estimate κ_{\max} with the Geršgorin circles over QQ^\dagger as

$$\kappa_{\max} \leq \max_i \sqrt{\left(\sum_j |[QQ^\dagger]_{i,j}| \right)}. \quad (\text{G6})$$

In conclusion, these bounds show that the trace distance is a meaningful quantity to bound the error on any other observable.

Appendix H: Details of the Krylov method

For the real-time TEBD algorithm we used the Krylov approximation to the propagated state for a local propagator on two sites. In this appendix, we clarify why sums throughout the algorithm can be done locally without involving the other parts of the subsystem. As an example we take the first step of the Gram-Schmidt orthogonalization procedure which can be generalized to sums. We denote the two local sites i and $j = i + 1$ acted on with $|C\rangle$ and the other parts of the system to left and right with $|L_i\rangle$ and $|R_j\rangle$. Moreover, the orthogonality center is in $|C\rangle$ which could be either site i or j . Using the Schmidt decomposition within the MPS, $|\psi\rangle$ can be written as

$$|v_1\rangle = |\psi\rangle = \sum_{i,j} |L_i\rangle |C_{i,j}\rangle |R_j\rangle \quad (\text{H1})$$

where the indices i and j run to the corresponding bond dimension at the splitting. In order to find the second basis vector $|v_2\rangle$ we calculate

$$|v_2\rangle \propto H_C |v_1\rangle - \langle v_1 | H_C^\dagger | v_1 \rangle |v_1\rangle, \quad (\text{H2})$$

with H_C the local Hamiltonian acting on the two sites of $|C\rangle$. Using Eq. (H1) to expand the second basis vector, we obtain the local summation:

$$|v_2\rangle \propto H_C \sum_{i,j} |L_i\rangle |C_{i,j}\rangle |R_j\rangle - \left[\sum_{i,i',j,j'} \langle L_{i'} | \langle C_{i',j'} | \langle R_{j'} | H_C^\dagger | L_i \rangle | C_{i,j} \rangle | R_j \rangle \right] \sum_{i,j} (|L_i\rangle |C_{i,j}\rangle |R_j\rangle). \quad (\text{H3})$$

The expression in brackets in Eq. (H3) is simplified due to the canonical form of the MPS. We obtain Kronecker deltas for $\langle L_{i'} | L_i \rangle = \delta_{i,i'}$ and $\langle R_{j'} | R_j \rangle = \delta_{j,j'}$. We point out that the expression $H_C |L_i\rangle |C_{i,j}\rangle |R_j\rangle$ is a short hand notation for $(\mathbb{I}_L \otimes H_C \otimes \mathbb{I}_R) |L_i\rangle \otimes |C_{i,j}\rangle \otimes |R_j\rangle = \mathbb{I}_L |L_i\rangle \otimes H_C |C_{i,j}\rangle \otimes \mathbb{I}_R |R_j\rangle$. These steps lead us to:

$$\begin{aligned} |v_2\rangle &= \sum_{i,j} |L_i\rangle (H_C |C_{i,j}\rangle) |R_j\rangle - \sum_{i,j} (|L_i\rangle |C_{i,j}\rangle |R_j\rangle) \sum_{i,j} \langle C_{i,j} | H_C |C_{i,j}\rangle \\ &= \sum_{i,j} |L_i\rangle (H_C |C_{i,j}\rangle) |R_j\rangle - \sum_{i,j} h |L_i\rangle |C_{i,j}\rangle |R_j\rangle, \end{aligned} \quad (\text{H4})$$

where we introduced the overlap $h \equiv \sum_{i,j} \langle C_{i,j} | H_C |C_{i,j}\rangle$ representing a scaling of the MPS. In order to scale the MPS with a scalar the orthogonality center is modified. The identity operator is acting $|L_i\rangle$ and $|R_j\rangle$ leaving them unchanged. Knowing that the orthogonalization procedure does not change any $|L_i\rangle$ and $|R_j\rangle$, a similar arguments apply to show that we can sum locally over the scaled Krylov basis $|v_k\rangle$.

We consider now the scaling of TEBD-Krylov in comparison to a usual TEBD taking the matrix exponential. In the usual TEBD implementation we can consider four steps: 1) Build two-site tensor, 2) calculate local propagator with Hamiltonian, 3) contract propagator to two site tensor, and 4) split two site tensor. Without considering speedup due to symmetries, a basic computational complexity analysis yields

$$\mathcal{O}_{\text{TEBD}} = \mathcal{O}(\chi^3 d^2) + \mathcal{O}(d^6) + \mathcal{O}(\chi^2 d^4) + \mathcal{O}(\chi^3 d^3), \quad (\text{H5})$$

where we assume cubic scaling for the matrix exponential and the splitting. In case of the Krylov-TEBD (KTEBD) we have the following steps: 1) contract the single site tensors to a two site tensor, 2) build n Krylov vectors, 3) m' calculations of overlap to previous vectors and m'' subtraction for orthogonalization ($m = m' + m''$), 4) taking the matrix exponential in the Krylov basis, 5) adding the weighted Krylov vectors to the solution, and 6) finally splitting the two site tensor. All the scalings are chronological, resulting in

$$\mathcal{O}_{\text{KTEBD}} = \mathcal{O}(\chi^3 d^2) + \mathcal{O}(n \chi^2 d^4) + \mathcal{O}(m \chi^2 d^2) + \mathcal{O}(n^3) + \mathcal{O}(n \chi^2 d^2) + \mathcal{O}(\chi^3 d^3). \quad (\text{H6})$$

We again assumed cubic scaling for matrix exponential and splitting, which is an upper bound in the case of the matrix exponential being tridiagonal and symmetric. The overall scaling seems to be dominated by the splitting with $\mathcal{O}(\chi^3 d^3)$, but we calculate the difference to see which algorithm is favorable in which cases:

$$\mathcal{O}_{\text{KTEBD}} - \mathcal{O}_{\text{TEBD}} = \mathcal{O}(n^3) - \mathcal{O}(d^6) + \mathcal{O}((n-1)\chi^2 d^4) + \mathcal{O}((m+n)\chi^2 d^2), \quad (\text{H7})$$

where TEBD is faster when the expression is greater than zero. If we assume in favor of KTEBD $d = \chi$ and $n = 2$, the second and the third term cancel each other. Thus, the remaining terms show that TEBD is always faster than KTEBD according to this scaling. These equations do not include quantum numbers. A careful study is contemplated to proof or disproof the scaling equations once both methods are implemented as pointed out in the future developments in Sec. VI.

Appendix I: Auxiliary calculations

The overlap between the truncated state $|\psi'\rangle$ and the untruncated normalized state $|\psi\rangle$ is defined over the singular values λ'_i of the truncated state and the singular values λ_i of the untruncated state.

In the following χ is the number of singular values in $|\psi'\rangle$ and χ_{\max} is the untruncated number of singular values. The singular values for the truncated state are calculated via a normalization

$$\lambda'_i = \frac{\lambda_i}{\sqrt{\sum_{j=1}^{\chi} \lambda_j^2}}. \quad (\text{I1})$$

This expression leads us directly to the overlap between the two states

$$\langle \psi' | \psi \rangle = \sum_{i=1}^{\chi_{\max}} \lambda'_i \lambda_i = \sum_{i=1}^{\chi} \lambda'_i \lambda_i = \frac{1}{\sqrt{\sum_{i=1}^{\chi} \lambda_i^2}} \sum_{i=1}^{\chi} \lambda_i^2 = \sqrt{\sum_{i=1}^{\chi} \lambda_i^2}. \quad (\text{I2})$$

For the error we calculate $1 - \langle \psi' | \psi \rangle$ and abbreviate with $x \equiv \langle \psi' | \psi \rangle$:

$$\begin{aligned} 1 - \langle \psi' | \psi \rangle &= 1 - x = \frac{1}{2} (1 - 2x + x^2) + \frac{1}{2} (1 - x^2) = \frac{1}{2} (1 - x)^2 + \frac{1}{2} (1 - x^2) \\ &\leq (1 - x^2) \end{aligned} \quad (\text{I3})$$

where the inequality is based on $x \leq 1$. By definition via Eq. (I2), x is real and bounded as $0 \leq x \leq 1$. This is based on the normalization constraint and knowing that the singular values λ_i are positive. The following inequality is only valid for $x \leq 1$: Equation (I3) is further simplified to the sum over the truncated singular values using the fact that the original state with χ_{\max} singular values was normalized:

$$1 - \langle \psi' | \psi \rangle \leq 1 - \sum_{i=1}^{\chi} \lambda_i^2 = 1 - \left(1 - \sum_{i=\chi+1}^{\chi_{\max}} \lambda_i^2 \right) = \sum_{i=\chi+1}^{\chi_{\max}} \lambda_i^2. \quad (\text{I4})$$

Appendix J: Files to reproduce plots in this work

We provide the files for the quantum Ising model, the long-range quantum Ising model and the dynamics of the Bose-Hubbard model as examples how python scripts for OSMPS are designed from the definition of the model to post-processing.

Listing 11. Example of nearest neighbor Ising model

```

1 import MPSPyLib as mps
2 import numpy as np
3 import sys
4 import os
5 import matplotlib.pyplot as plt
6 from mpl_toolkits.mplot3d.axes3d import Axes3D
7 from matplotlib import cm
8 from scipy.optimize import curve_fit
9 from copy import deepcopy
10
11
12 def main(PostProcess):
13     """
```

```

14 Main method running simulation for the following plots in the
    OSMPS paper:
15
16 1) Bond entropy of the Ising model as function of the external
    field and
17 the system size (see Figure 2).
18 2) Finite size scaling to find the critical value in the
    thermodynamic
19 limit (see Figure 4).
20 3) The scaling of resources in Figures 16(c) and (d).
21
22 **Arguments**
23
24 PostProcess : Boolean (here)
25 When PostProcess is ``True``, then data for the level is
    evaluated.
26 Otherwise simulation is executed (with MPI). In order to
    pass arguments
27 on the command line use ``T`` for ``True`` and ``F`` for ``
    False``.
28
29 # Build spin operators
30 # -----
31
32 # These are rotated Pauli operators to obtain a diagonal
    generator for Z2
33 Operators = mps.BuildSpinOperators(spin=0.5)
34 Operators['sx'] = 2 * Operators['sz']
35 Operators['sz'] = - (Operators['splus'] + Operators['sminus'])
36 Operators['gen'] = np.array([[0, 0], [0, 1]])
37
38 # Define Hamiltonian / system settings
39 # -----
40
41 H = mps.MPO(Operators)
42 H.AddMPOTerm('bond', ['sz', 'sz'], hparam='J', weight=-1.0)
43 H.AddMPOTerm('site', 'sx', hparam='h', weight=-1.0)
44
45 # overall energy
46 J = 1.0
47
48 # Choose length of the systems for equal spacing in log-plot
49 # (reverse order to submit long simulations first)
50 nl = 30
51 Llist = list(map(int, 10*(np.linspace(0.65, 3.30103, nl))))
    [::-1]

```

```

52
53     # Grid of the external field
54     nh = 51
55     hlist = np.linspace(0.5, 1.5, nh)
56
57     # Define Observables
58     # -----
59
60     # Initialize instance of observable class and add local
        observable
61     myObservables = mps.Observables(Operators)
62     myObservables.AddObservable('site', 'sz', 'z')
63
64     # add correlation functions
65     myObservables.AddObservable('corr', ['sz', 'sz'], 'zz')
66
67     # Specify convergence parameters
68     # -----
69
70     nc = 2
71     conv = mps.MPSConvParam(max_bond_dimension=40, variance_tol=1e
        -10,
72                             local_tol=1e-10, max_num_sweeps=4)
73     conv.AddModifiedConvergenceParameters(0, ['max_bond_dimension',
74                                               'max_num_sweeps'],
75                                           [80, 4])
76
77     # Create list of simulations
78     # -----
79
80     params = []
81
82     for jj in range(nl):
83         ll = Llist[jj]
84
85         for hh in hlist:
86
87             params.append({
88                 'simtype' : 'Finite',
89                 # Filenames and directories
90                 'job_ID' : 'sim_01_ising_z2_
91                     ',
92                 'unique_ID' : '_L%04d'%ll + '_h
93                     %3.6F'%hh,
94                 'Write_Directory' : 'TMP_01_ISING/',
95                 'Output_Directory' : 'OUTPUTS_01_ISING

```

```

    '/',
93     # System size and Hamiltonian parameters
94     'L' : ll,
95     'J' : J,
96     'h' : hh,
97     # Other parameters
98     'MPSConvergenceParameters' : conv,
99     'MPSObservables' : myObservables,
100    # For error calculation (gap)
101    'n_excited_states' : 1,
102    'eMPSConvergenceParameters' : conv,
103    'logfile' : True,
104    'verbose' : 1,
105    # Z2 symmetry
106    'Discrete_generators' : ['gen'],
107    'Discrete_quantum_numbers' : [0]
108    })
109
110    if(not PostProcess):
111        # Run simulations
112        # -----
113
114        # Generate sbatch script for CSM cluster
115        MainFiles = mps.WriteMPSParallelFiles(params, Operators, H,
116                                              {'queueing' : 'slurm'
117                                              ,
118                                              'partition' : 'lcarr'
119                                              ,
120                                              'time' : '143:59:59'
121                                              ,
122                                              'nodes' : ['062', '063', '064'],
123                                              'ThisFileName' : '01_Ising.py',
124                                              'mpi' : 'srun'},
125                                              PostProcess=
126                                              PostProcess)
127
128        return
129
130    # Evaluation of the simulations
131    # =====
132
133    base = 'OUTPUTS_01_ISING/'
134
135    if(os.path.isfile(base + "01_Ising_BondEntropy.npy")):

```

```

132     # Read from previous PostProcess
133     bond_entropy = np.load(base + "01_Ising_BondEntropy.npy")
134     err_bentropy = np.load(base + "01_Ising_BondEntropyError.
        npy")
135     converged = np.load(base + "01_Ising_converged.npy")
136     variance = np.load(base + "01_Ising_variance.npy")
137     runtimes = np.load(base + "01_Ising_runtimes.npy")
138     filesize = np.load(base + "01_Ising_filesize.npy")
139     bonddim2 = np.load(base + "01_Ising_bonddim2.npy")
140 else:
141     # Read from Fortran output
142     MainFiles = mps.WriteFiles(params, Operators, H,
143                               PostProcess=True)
144     Outputs = mps.ReadStaticObservables(params)
145
146     # Create array for values of bond entropy
147     bond_entropy = np.zeros((nl, nh, nc), dtype=float)
148     err_bentropy = np.zeros((nl, nh, nc), dtype=float)
149     converged = np.zeros((nl, nh, nc))
150     variance = np.zeros((nl, nh, nc))
151     runtimes = np.zeros((nl, nh, nc))
152     filesize = np.zeros((nl, nh))
153     bonddim2 = np.zeros((nl, nh))
154
155     # Get mapping from simulation to hashes
156     hdic = read_hashes(params[0])
157
158     # index for Outputs-list
159     idx = 0
160
161     # Looping over the results
162     for ii in range(nl):
163         for jj in range(nh):
164             # idx      : ground state, convergence parameters 0
165             # idx + 1 : ground state, convergence parameters 1
166             # idx + 2 : excited state, convergence parameters 0
167             # idx + 3 : excited state, convergence parameters 1
168
169             for kk in range(nc):
170                 Output = Outputs[idx + kk]
171                 eOutput = Outputs[idx + kk + 2]
172
173                 bond_entropy[ii, jj, kk] = Output['BondEntropy'
174                                                    ][Llist[ii] // 2]
175                 err_bentropy[ii, jj, kk] = error_entropy(Output
176                                                            , eOutput)

```



```

175         converged[ii, jj, kk] = int(Output["converged"]
176             == "T")
177         variance[ii, jj, kk] = Output["variance"]
178         runtimes[ii, jj, kk] = mps.get_runtime(Output)
179
180         out = Outputs[idx]['Output_Directory']
181         key = out + Outputs[idx]['job_ID'] + Outputs[idx]['
182             unique_ID']
183         thishash = hdic[key]
184
185         filesize[ii, jj] = os.path.getsize(out + thishash +
186             '_002.bin') / 1024.**2
187         bonddim2[ii, jj] = Outputs[idx + 1]['bond_dimension
188             ']
189
190         idx += 4
191
192         np.save(base + "01_Ising_BondEntropy.npy", bond_entropy)
193         np.save(base + "01_Ising_BondEntropyError.npy",
194             err_bentropy)
195         np.save(base + "01_Ising_converged.npy", converged)
196         np.save(base + "01_Ising_variance.npy", variance)
197         np.save(base + "01_Ising_runtimes.npy", runtimes)
198         np.save(base + "01_Ising_filesize.npy", filesize)
199         np.save(base + "01_Ising_bonddim2.npy", bonddim2)
200
201         # Use tex fonts throughout
202         plt.rc('text', usetex=True)
203         plt.rc('font', family='serif', size=22)
204
205         # Surface plot for bond entropy
206         # -----
207
208         # specify which convergence parameter
209         ncp = 1
210         surface_plot_bond_entropy(np.array(Llist), hlist, bond_entropy
211            [:, :, ncp])
212
213         # Set upper and lower bound for plotting manually
214         err_bentropy[np.isnan(err_bentropy)] = 1.0
215         err_bentropy[err_bentropy < 1e-7] = 1e-7
216         err_bentropy[err_bentropy > 1.0] = 1.0
217         #surface_errplot_bond_entropy(np.array(Llist), hlist,
218             err_bentropy[:, :, ncp])
219
220         # Plot scaling resources

```

```

214 # -----
215
216 plot_scaling_filesize(np.array(Llist), filesize[:, 25],
217                        bonddim2[:, 25])
218
219 # Finite size scaling for bond entropy (exclude very small
220 # systems)
221 # -----
222
223 # Find the maximum for each systems size
224 hc = np.zeros((nl), dtype=float)
225
226 for ii in range(nl):
227     pos = np.argmax(bond_entropy[ii, :, ncp])
228     hc[ii] = hlist[pos]
229
230 p0 = [1.0, 1.0]
231 [hci, nu], covar = curve_fit(fit_func, Llist[:20], hc[:20], p0=
232                             p0)
233
234 fig = plt.figure()
235 ax = fig.add_subplot(111)
236 ax.plot(Llist[:20], hc[:20], 'go-', label=r'$h_c(L)$')
237 ax.plot(Llist[:20], [hci] * len(hc[:20]), 'b-', label=r'$h_c =
238             %3.4f$ (FSS)'%(hci))
239 ax.plot(Llist[:20], [hci - covar[0,0]] * len(hc[:20]), 'b--')
240 ax.plot(Llist[:20], [hci + covar[0,0]] * len(hc[:20]), 'b--')
241 ax.set_xlabel(r'\textbf{System size} $L$')
242 ax.set_ylabel(r'\textbf{Critical field} $h_c$')
243 ax.set_xscale('log')
244
245 plt.legend(loc='lower right')
246 plt.savefig("01_Ising_FSS.pdf")
247
248 return
249
250 def fit_func(ll, hc, nu):
251     """
252     Fitting functions for the finite size scaling.
253
254     **Arguments**
255
256     ll : int

```

```

255         System size.
256
257     hc : float
258         Value for the critical point to be fitted.
259
260     nu : float
261         Value of the critical exponent to be fitted.
262     """
263     return hc - 11**(-1 / nu)
264
265
266 def error_entropy(Out0, Out1):
267     """
268     Calculate the error of the bond entropy of the ground state
269     with the
270     results of the ground state and first excited state.
271
272     **Arguments**
273
274     Out0 : dictionary
275         containing the results of the ground state as dictionary.
276
277     Out1 : dictionary
278         containing the results of the first excited state as
279         dictionary.
280     """
281     gap = Out1['energy'] - Out0['energy']
282     coeff = np.sqrt(2 * Out0['variance']) / gap
283     dim = Out0['L'] // 2
284
285     if(coeff > 1 / np.exp(1)): return np.nan
286
287     return coeff * (dim * np.log(2) - np.log(coeff))
288
289
290 def surface_plot_bond_entropy(xgrid, ygrid, data):
291     """
292     A general 3d plot used to plot the bond entropy.
293
294     **Arguments**
295
296     xgrid : numpy array (1D)
297         Contains the data points for the first axis.
298         Log10 is applied to this axis.
299
300     ygrid : numpy array (1D)

```

```

299         Contains the data points for the second axis.
300
301     data : numpy 2d array
302         Contains the values of the z-axis.
303     """
304     # Meshgrid (logrithmic over system size)
305     xm, ym = np.meshgrid(np.log10(xgrid), ygrid)
306
307     fig = plt.figure()
308     ax = fig.add_subplot(111, projection='3d')
309     surf = ax.plot_surface(xm, ym, np.transpose(data),
310                           rstride=1, cstride=1,
311                           cmap=cm.jet,
312                           linewidth=0, antialiased=False)
313
314     ax.set_xlabel(r'\textbf{Size} $\log_{10}(L)$', labelpad=20.0)
315     ax.set_ylabel(r'\textbf{External field} $h$', labelpad=15.0)
316     ax.set_zlabel(r'\textbf{Bond entropy} $$$', labelpad=10.0)
317     ax.view_init(elev=35, azimuth=-160)
318
319     cbar = plt.colorbar(surf, shrink=0.8)
320     cbar.set_label(r'\textbf{Bond entropy} $$$')
321
322     plt.tight_layout(pad=1.5)
323     plt.savefig('01_Ising_BondEntropy.pdf')
324
325     return
326
327
328 def surface_errrplot_bond_entropy(xgrid, ygrid, data):
329     """
330     A general 3d plot used to plot the error of the bond entropy.
331
332     **Arguments**
333
334     xgrid : numpy array (1D)
335         Contains the data points for the first axis.
336         Log10 is applied to this axis.
337
338     ygrid : numpy array (1D)
339         Contains the data points for the second axis.
340
341     data : numpy 2d array
342         Contains the values of the z-axis. Log10 is applied
343         to these values.
344     """

```

```

345     # Meshgrid (logarithmic over system size)
346     xm, ym = np.meshgrid(np.log10(xgrid), ygrid)
347
348     fig = plt.figure()
349     ax = fig.add_subplot(111, projection='3d')
350     surf = ax.plot_surface(xm, ym, np.transpose(np.log10(data)),
351                           rstride=1, cstride=1,
352                           cmap=cm.jet,
353                           linewidth=0, antialiased=False)
354
355     ax.set_xlabel(r'\textbf{System size} $\log_{10}(L)$', labelpad
356                  =20.0)
357     ax.set_ylabel(r'\textbf{External field} $h$', labelpad=15.0)
358     ax.set_zlabel(r'\textbf{Error bound bond entropy} $\log_{10}(\epsilon_S)$', labelpad=10.0)
359
360     cbar = plt.colorbar(surf)
361     cbar.set_label(r'\textbf{Error bound bond entropy} $\log_{10}(\epsilon_S)$')
362
363     plt.savefig("01_Ising_ErrBondEntropy.pdf", bbox_inches="tight")
364
365     return
366
367 def plot_scaling_filesize(Ls, filesize, bonddims):
368     """
369     Plot the filesize of a ground state as a function of the
370     system size.
371
372     **Arguments**
373
374     Ls : list
375         Contains the list of different system sizes.
376
377     filesize : list
378         Contains the corresponding filesize at each
379         system size.
380
381     bonddims : list
382         Contains the information about the maximal
383         bond dimension used for a simulation.
384     """
385
386     fig = plt.figure()

```

```

387     ax = fig.add_subplot(111)
388     ax.plot(Ls, filesize, 'bo-')
389     for elem in ax.get_yticklabels():
390         elem.set_color('b')
391
392     axo = ax.twinx()
393     axo.plot(Ls, bonddims, 'ro--')
394     for elem in axo.get_yticklabels():
395         elem.set_color('r')
396     axo.set_ylim([0, 82])
397
398     ax.set_xlabel(r'\textbf{System size} $L$')
399     ax.tick_params(direction='out', pad=2)
400
401     ax.set_ylabel(r'\textbf{File size} $S_{\mathrm{MB}}$', color='b')
402     axo.set_ylabel(r'\textbf{Bond dimension} $\chi$', color='r')
403
404     plt.tight_layout(pad=0.2)
405     plt.savefig('C2_ScalingFile.pdf')
406
407     return
408
409
410 def plot_scaling_runtimes(Ls, runtimes, bonddims):
411     """
412     Plot the runtimes as a function of the system size.
413
414     **Arguments**
415
416     Ls : list
417         Contains the list of different system sizes.
418
419     runtimes : list
420         Contains the corresponding runtimes at each
421         system size.
422
423     bonddims : list
424         Contains the information about the maximal
425         bond dimension used for a simulation.
426     """
427     fig = plt.figure()
428     ax = fig.add_subplot(111)
429     ax.plot(Ls, runtimes, 'bo-')
430     for elem in ax.get_yticklabels():
431         elem.set_color('b')

```

```

432
433     axo = ax.twinx()
434     axo.plot(Ls, bonddims, 'ro--')
435     for elem in axo.get_yticklabels():
436         elem.set_color('r')
437     axo.set_ylim([0, 82])
438
439     ax.set_xlabel(r'\textbf{System size} $L$')
440     ax.tick_params(direction='out', pad=2)
441
442     ax.set_ylabel(r'\textbf{CPU time} $T_{\mathrm{CPU}}$ / \mathrm{s}
443                  $'$, color='b')
444     axo.set_ylabel(r'\textbf{Bond dimension} $\chi$', color='r')
445
446     plt.tight_layout(pad=0.2)
447     plt.savefig('C2_ScalingTime.pdf')
448
449     return
450
451 def read_hashes(param):
452     """
453     Return mapping from names to hash to read file size.
454
455     **Arguments**
456
457     param : dict
458         example dictionary to get name of output folder etc.
459     """
460     out = param['Output_Directory']
461     jobid = param['job_ID']
462
463     fh = open(out + jobid + '_static_mapping.dat', 'r')
464     dic = {}
465
466     for line in fh:
467         key, val = line.replace('\n', '').split()
468         dic[key] = val
469
470     return dic
471
472
473 if(__name__ == "__main__"):
474     args = {"PostProcess" : "F"}
475
476     for elem in sys.argv[1:]:

```



```

477     try:
478         key, val = elem.replace("--", "").split("=")
479         args[key] = val
480     except:
481         pass
482
483     args["PostProcess"] = ((args["PostProcess"] == "T") or
484                           (args["PostProcess"] == "True"))
485     main(args["PostProcess"])

```

Listing 12. Example of nearest neighbor Ising model

```

1  import MPSPyLib as mps
2  import numpy as np
3  import sys
4  import os
5  import matplotlib.pyplot as plt
6  from mpl_toolkits.mplot3d.axes3d import Axes3D
7  from matplotlib import cm
8  from copy import deepcopy
9
10
11 def main(PostProcess):
12     """
13     Main method running simulation to find excited states for the
14     long range
15     quantum Ising model. Either running simulations with MPI or
16     doing evaluation
17     on a single core. It produces the following plot, but does not
18     contain the
19     plots for the error analysis against the simulations with Z2
20     symmetry.
21
22     1) Energy gap for the long-range Ising model, see Figure 8(a).
23
24     **Arguments**
25
26     PostProcess : Boolean (here)
27         When PostProcess is ``True``, then data for the level is
28         evaluated.
29         Otherwise simulation is executed (with MPI). In order to
30         pass arguments
31         on the command line use ``T`` for ``True`` and ``F`` for ``
32         False``.
33
34     """
35     # overall energy
36     J = 1.0

```

```

29
30     # Choose length of the systems / number of excited states
31     ll = 128
32     nl = 1
33     ne = 4
34
35     # Grid of the external field / interaction strength or decay
36     nh = 25
37     hlist = np.linspace(0.7, 1.7, nh)
38
39     nalpha = 25
40     alphaslist = np.linspace(2.0, 4.0, nalpha)
41
42     # Build operators (start with spin operators, get sigma_{x,z})
43     # -----
44
45     Operators = mps.BuildSpinOperators(spin=0.5)
46     Operators['sx'] = Operators['splus'] + Operators['sminus']
47     Operators['sz'] *= 2
48
49     # Define Observables
50     # -----
51
52     myObservables = mps.Observables(Operators)
53
54     # Site terms
55     myObservables.AddObservable('site', 'sz', 'z')
56
57     # correlation functions
58     myObservables.AddObservable('corr', ['sz', 'sz'], 'zz')
59     myObservables.SpecifyCorrelationRange(1000)
60
61     # Specify convergence parameters
62     # -----
63
64     nc = 2
65     conv = mps.MPSConvParam(max_bond_dimension=40,
66                             variance_tol=1e-10,
67                             local_tol=1e-10,
68                             max_num_sweeps=4)
69     conv.AddModifiedConvergenceParameters(0, ['max_bond_dimension',
70         'max_num_sweeps'], [80, 4])
71
72     # Create list of simulations and Hamiltonians
73     # -----

```

```

74     params = []
75     Hlist = []
76
77     for jj in range(nalpha):
78         alpha = alphalist[jj]
79
80         # Define the Hamiltonian
81         # -----
82
83         H = mps.MPO(Operators, PostProcess=PostProcess)
84         invalpha = lambda x: 1/(x**alpha)
85         H.AddMPOTerm('InfiniteFunction', ['sz', 'sz'],
86                     hparam='J', weight=-1.0, func=invalpha,
87                     L=11, tol=1e-10)
88         H.AddMPOTerm('site', 'sx', hparam='h', weight=-1.0)
89
90     for hh in hlist:
91
92         Hlist.append(deepcopy(H))
93
94         params.append({
95             'simtype'                        : 'Finite',
96             # Filenames and directories
97             'job_ID'                         : 'sim_02_lrising_L
98                 %04d'%11,
99             'unique_ID'                     : '_alpha%3.6F'%
100                 alpha + \
101                 '_h%3.6F'%hh,
102             'Write_Directory'               : 'TMP_02_LRISING/'
103             ,
104             'Output_Directory'              : '
105                 OUTPUTS_02_LRISING/' ,
106             # System size and Hamiltonian parameters
107             'L'                             : 11,
108             'J'                             : J,
109             'h'                             : hh,
110             'alpha'                         : alpha,
111             # Other parameters
112             'MPSConvergenceParameters'     : conv,
113             'MPSObservables'               : myObservables,
114             # eMPS
115             'n_excited_states'              : ne,
116             'eMPSConvergenceParameters'   : conv,
117             'eMPSObservables'              : myObservables
118         })

```

```

116     if(not PostProcess):
117         # Generate sbatch script for CSM cluster
118         # -----
119
120         MainFiles = mps.WriteMPSParallelFiles(params, Operators,
121             Hlist,
122
123             {'queueing' : 'slurm'
124             ,
125             'partition' : 'lcarr'
126             ,
127             'time' : '143:59:59'
128             ,
129             'nodes' : ['084', '
130                        085', '086', '087
131                        '],
132             'ThisFileName' : '02
133                        _LRIsing.py',
134             'mpi' : 'srun'},
135         PostProcess=
136             PostProcess)
137
138     return
139
140     # Evaluation of the simulations
141     # =====
142
143     base = 'OUTPUTS_02_LRISING/'
144
145     if(os.path.isfile(base + "02_LRIsing_Energies.npy")):
146         energies = np.load(base + "02_LRIsing_Energies.npy")
147     else:
148         MainFiles = mps.WriteFiles(params, Operators, Hlist,
149             PostProcess=True)
150         Outputs = mps.ReadStaticObservables(params)
151
152         # Create array for values of bond entropy
153         energies = np.zeros((nl, nh, nalpna, ne + 1, nc), dtype=
154             float)
155
156         # index for Outputs-list
157         idx = 0
158
159         # Looping over the results
160         for ii in range(nl):
161             for jj in range(nh):
162                 for kk in range(nalpna):

```

```

153         for ee in range(ne + 1):
154             for cc in range(nc):
155                 Output = Outputs[idx]
156                 if(cc == 0): print(Output['energy'], ee
157                                , Output['state'],
158                                Output['
                                convergence_parameter
                                '])
159                 energies[ii, jj, kk, ee, cc] = Output['
160                 energy']
161
162                 idx += 1
163
164     np.save(base + "02_LRIsing_Energies.npy", energies)
165
166     # Sort energies if flag
167     sort = True
168     if(sort):
169         for ii in range(nl):
170             for jj in range(nh):
171                 for kk in range(nalpha):
172                     for cc in range(nc):
173                         idx = energies[ii, jj, kk, :, cc].argsort()
174                         energies[ii, jj, kk, :, cc] = energies[ii,
175                         jj, kk, idx, cc]
176
177     # Specify plotted convergence parameters
178     pnc = nc - 1
179
180     diff01 = energies[0, :, :, 1, pnc] - energies[0, :, :, 0, pnc]
181     diff02 = energies[0, :, :, 2, pnc] - energies[0, :, :, 0, pnc]
182     diff12 = energies[0, :, :, 2, pnc] - energies[0, :, :, 1, pnc]
183
184     # Surface plot for bond entropy
185     # -----
186
187     # Meshgrid (logarithmic over system size)
188     xm, ym = np.meshgrid(hlist, alphalist)
189
190     # Use tex fonts
191     plt.rc('text', usetex=True)
192     plt.rc('font', family='serif', size=22)
193
194     plot_gaps_surface(xm, ym, diff01, diff02)
195
196     return

```

```

194
195
196 def plot_gaps_surface(xm, ym, diff01, diff02):
197     fig = plt.figure()
198     ax = fig.add_subplot(111, projection='3d')
199
200     vmin = min(np.min(diff01), np.min(diff02))
201     vmax = max(np.max(diff01), np.max(diff02))
202
203     surf01 = ax.plot_surface(xm, ym, np.transpose(diff01),
204                             rstride=1, cstride=1, vmin=vmin, vmax=
205                                 vmax,
206                                 cmap=cm.jet, #coolwarm,
207                                 linewidth=0, antialiased=False)
208     surf02 = ax.plot_surface(xm, ym, np.transpose(diff02),
209                             rstride=1, cstride=1, vmin=vmin, vmax=
210                                 vmax,
211                                 cmap=cm.jet, #coolwarm,
212                                 linewidth=0, antialiased=False)
213
214     ax.set_xlabel(r'\textbf{External field}  $h$ ', labelpad=15.0)
215     ax.set_ylabel(r'\textbf{Power law decay}  $\alpha$ ', labelpad
216                 =15.0)
217     ax.set_zlabel(r'\textbf{Energy gap}  $\Delta E$ ', labelpad=10.0)
218
219     cbar = plt.colorbar(surf01, shrink=0.8, pad=0.07)
220
221     plt.tight_layout(pad=2.5)
222     plt.savefig('02_LRIsing_Gaps.pdf')
223
224     return
225
226
227 if(__name__ == "__main__"):
228     args = {"PostProcess" : "F"}
229
230     for elem in sys.argv[1:]:
231         try:
232             key, val = elem.replace("--", "").split("=")
233             args[key] = val
234         except:
235             pass
236
237     args["PostProcess"] = ((args["PostProcess"] == "T") or
238                           (args["PostProcess"] == "True"))
239     main(args["PostProcess"])

```

Listing 13. Example of nearest neighbor Ising model

```

1  import MPSPyLib as mps
2  import numpy as np
3  import sys
4  import os.path
5  import matplotlib.pyplot as plt
6  from matplotlib import cm
7
8
9  def main(PostProcess):
10     """
11     Main method to simulate the Bose-Hubbard model in a rotating
12     saddle
13     point potential. The script reproduces Figure 11(b).
14
15     **Arguments**
16
17     PostProcess : Boolean (here)
18         When PostProcess is ``True``, then data for the level is
19         evaluated.
20         Otherwise simulation is executed (with MPI). In order to
21         pass arguments
22         on the command line use ``T`` for ``True`` and ``F`` for ``
23         False``.
24     """
25     # Build operators
26     # -----
27
28     Operators = mps.BuildBoseOperators(6)
29     Operators['interaction'] = 0.5 * (np.dot(Operators['nbttotal'],
30                                             Operators['nbttotal'])
31                                     - Operators['nbttotal'])
32
33     # Define Hamiltonian / system settings
34     # -----
35
36     # system size and filling
37     L = 30
38     fill = 20
39
40     H = mps.MPO(Operators)
41     H.AddMPOTerm('bond', ['bdagger', 'b'], hparam='t', weight=-1.0)
42     H.AddMPOTerm('site', 'interaction', hparam='U', weight=1.0)
43     H.AddMPOTerm('site', 'nbttotal', hparam='Vt', weight=1.0)

```



```

40
41     # accelaration of rotating potential
42     # (t=100 39.8 rotations, t=101: 40.6 rotations)
43     alpha = 0.05
44
45     # and scaling prefactor  $c * (x^{**2} - y^{**2})$ 
46     c = 0.01
47
48     # overall energy scale
49     U = 1.0
50
51     # choose different tunneling strengths
52     nt = 11
53     ts = np.linspace(0.0, 0.5, nt)
54
55     # Define observables
56     # -----
57
58     myObs = mps.Observables(Operators)
59     myObs.AddObservable('site', 'nbttotal', 'n')
60     myObs.AddObservable('corr', ['b', 'bdagger'], 'spdm')
61
62     # Specify convergence parameters
63     # -----
64
65     conv = mps.MPSConvParam(max_bond_dimension=60)
66
67     # Define quench function
68     def Vt(t, c=c, alpha=alpha, L=L):
69         # Grid with particles symmetric around 0 at unit distance
70         grid = np.linspace(-L / 2 + 0.5, L / 2 - 0.5, L)
71
72         phit = np.pi / 4 + 0.5 * alpha * t**2
73
74         return c * ((grid * np.cos(phit))**2 - (grid * np.sin(phit)
75                     )**2)
76
77     tconv = mps.TEBDConvParam(max_bond_dimension=60)
78     Quench = mps.QuenchList(H)
79     Quench.AddQuench(['Vt'], 100.0, 0.01, [Vt], stepsforoutput=10,
80                     ConvergenceParameters=tconv)
81
82     params = []
83     for ii in range(nt):
84         # Get tunneling parameters
85         tt = ts[ii]

```

```

85
86     params.append({
87         'simtype' : 'Finite',
88         # Directories
89         'job_ID' : 'BoseHubbard_RotSaddle',
90         'unique_ID' : '_tau%3.6f'%(tt),
91         'Write_Directory' : 'TMP_03_Hubbard/',
92         'Output_Directory' : 'OUTPUTS_03_Hubbard/',
93         # System parameters
94         'L' : L,
95         'U' : U,
96         't' : tt,
97         'Vt' : np.zeros((L)),
98         'MPSConvergenceParameters' : conv,
99         'Abelian_generators' : ['nbtotall'],
100        'Abelian_quantum_numbers' : [fill],
101        'MPSObservables' : myObs,
102        'Quenches' : Quench,
103        'DynamicsObservables' : myObs,
104        # Other settings
105        'logfile' : True
106    })
107
108    if(not PostProcess):
109        # Run simulations
110        # -----
111
112        MainFiles = mps.WriteMPSParallelFiles(params, Operators, H,
113                                                {'queueing' : 'slurm'
114                                                ,
115                                                'partition' : 'lcarr'
116                                                ,
117                                                'time' : '143:59:59'
118                                                ,
119                                                'nodes' : ['026', '
120                                                027'],
121                                                'ThisFileName' : '05
122                                                _RotSaddle.py',
123                                                'mpi' : 'srun'}),
124        PostProcess=
125            PostProcess)
126
127    return
128
129    # Post Process
130    # -----
131
132

```

```

125     # Number of measurements in time evolution
126     nm = 1000
127
128     if(os.path.isfile("OUTPUTS_03_Hubbard/nstdxt.npy")):
129         # Data has been saved in a previous post-processing
130         nstdxt = np.load("OUTPUTS_03_Hubbard/nstdxt.npy")
131     else:
132         # Read data from Fortran output
133         MainFiles = mps.WriteFiles(params, Operators, H,
134                                     PostProcess=PostProcess)
135         Outputs = mps.ReadDynamicObservables(params)
136
137         # Array to save standard deviation on particle number over
138         # space for each t
139         nstdx = np.zeros((nt, nm))
140
141         for ii in range(nt):
142             for tt in range(nm):
143                 nstdx[ii, tt] = np.std(Outputs[ii][tt]['n'])
144
145         # Get a std deviation in time for 50 subsequent
146         # measurements
147         nstdxt = np.zeros((nt, nm - 50))
148
149         for ii in range(nt):
150             for jj in range(50, nm):
151                 nstdxt[ii, jj - 50] = np.std(nstdx[ii, jj - 50:jj])
152
153         np.save("OUTPUTS_03_Hubbard/nstdxt.npy", nstdxt)
154
155     # Plotting
156     # -----
157
158     plt.rc('text', usetex=True)
159     plt.rc('font', family='serif', size=22)
160
161     color = cm.rainbow(np.linspace(0, 1, ts.shape[0]))
162     mi = np.min(np.array(ts))
163     ma = np.max(np.array(ts))
164     clabel = r'\textbf{Tunneling} $\mathcal{J}$'
165     cticks = np.linspace(mi, ma, 6)
166     cbarinfo = plt.contourf([[0, 0], [0, 0]],
167                             np.linspace(mi, ma, 101), cmap=cm.
168                                     rainbow)
169
170     plt.clf()

```

```

168     fig = plt.figure()
169     ax = fig.add_subplot(111)
170
171     for ii in range(nt):
172         ax.plot(np.linspace(0.5, 100.0, nm - 50), nstdxt[ii], c=
            color[ii])
173
174     cbar = plt.colorbar(cbarinfo)
175     cbar.set_ticks(cticks)
176     cbar.set_label(clabel)
177
178     ax.set_xlabel(r'\textbf{Time} $t$')
179     ax.set_ylabel(r'\textbf{Standard deviation} $\sigma_t$')
180
181     plt.tight_layout(pad=0.2)
182     plt.savefig('05_RotSaddle_std_txn.pdf')
183
184     return
185
186
187 if(__name__ == "__main__"):
188     args = {"PostProcess" : "F"}
189
190     for elem in sys.argv[1:]:
191         try:
192             key, val = elem.replace("--", "").split("=")
193             args[key] = val
194         except:
195             pass
196
197     args["PostProcess"] = ((args["PostProcess"] == "T") or
198                           (args["PostProcess"] == "True"))
199     main(args["PostProcess"])

```
